

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

DISEÑO Y CONSTRUCCIÓN DE UNA SILLA DE RUEDAS ELÉCTRICA DE BAJO COSTE



Memoria y Anexos

Autor:	Adrián Mora Sardiña
Director:	Sebastián Tornil Sin
Co-Director:	Juan Andrade Cetto
Convocatoria:	Mayo 2018

Resum

El següent projecte consisteix en el disseny, la construcció i el control d'una cadira de rodes elèctrica de baix cost per a persones amb discapacitats físiques avançades i pocs recursos econòmics.

Aquest treball està pensat com una primera fase del projecte "*Diseño y construcción de una silla exploradora inteligente de bajo coste*", a on es vol dotar a la cadira de rodes elèctrica d'un sistema de localització i navegació per a què es pugui moure de forma autònoma, específicament en entorns interiors, el qual es desenvoluparà a l'Institut de Robòtica i Informàtica Industrial CSIC-UPC.

El sistema que proporciona moviment a la cadira de rodes elèctrica s'ha dissenyat a partir d'un vehicle de bateria portàtil i recarregable, en el qual s'ha realitzat un estudi d'enginyeria inversa per així poder aprofitar al màxim els components, tant electrònics com mecànics, que ja posseeix de sèrie, per tal de reduir els costos del projecte.

Finalment, la velocitat de la cadira es controla a través d'un joystick que, connectat a un ordinador de placa reduïda, envia les comandes necessàries de velocitat a la targeta controladora principal de la cadira.

Resumen

El siguiente proyecto consiste en el diseño, la construcción y el control de una silla de ruedas eléctrica de bajo coste para personas con discapacidades físicas avanzadas y pocos recursos económicos.

Este trabajo está pensado como una primera fase del proyecto *"Diseño y construcción de una silla exploradora inteligente de bajo coste"*, donde se quiere dotar a la silla de ruedas eléctrica de un sistema de localización y navegación para que pueda moverse de forma autónoma, específicamente en entornos interiores, el cual se desarrollará en el Institut de Robòtica i Informàtica Industrial CSIC-UPC.

El sistema que proporciona movimiento a la silla de ruedas se ha diseñado a partir de un vehículo de batería portátil y recargable, en el cual se ha realizado un estudio de ingeniería inversa para así poder aprovechar al máximo los componentes, tanto electrónicos como mecánicos, que ya posee de serie, con tal de reducir los costes del proyecto.

Finalmente, la velocidad de la silla se controla a través de un joystick que, conectado a un ordenador de placa reducida, envía los comandos necesarios de velocidad a la tarjeta controladora principal de la silla.

Abstract

This project consists in the design, construction and control of a low cost electric wheelchair for people with advanced physical disabilities and low-income.

This work is intended to be the first stage of the project "*Diseño y construcción de una silla exploradora inteligente de bajo coste*", in which it is planned to add a navigation and a localization system to the electric wheelchair so it can move indoors in an autonomous and securely way. This project will be later developed in the Institut de Robòtica i Informàtica Industrial CSIC-UPC.

The system which provides movement to the electric wheelchair has been designed from a portable rechargeable battery vehicle, and performing a reverse engineering procedure to it, it has been possible to use all their electronic and mechanic components in order to reduce the project costs.

The speed of the wheelchair is controlled using a joystick which, connected to a single board computer, sends the necessary velocity commands to the main controller board of the wheelchair.



Agradecimientos

Agradecer primeramente al director del Institut de Robòtica i Informàtica Industrial CSIC-UPC Juan Andrade Cetto, por darme la oportunidad de desarrollar su propuesta de Trabajo de Fin de Grado en el IRI, donde he aprendido muchísimo a lo largo del proyecto.

Dar las gracias también a los compañeros del laboratorio de robótica móvil en general, y en especial a Sergi Hernández, Patrick Grosch, Fernando Herrero y Alejandro López por el soporte técnico y el apoyo moral. Son fantásticos tanto profesionalmente como personalmente.

A Vladimir Hurtado, por haber sido un gran compañero de soporte y con el que he aprendido mucho y he compartido buenos momentos.

Finalmente, agradecer a mi familia el apoyo que he obtenido de todos ellos a lo largo de mi carrera.

Muchas gracias a todos.





Índice

RESUM	I
RESUMEN	II
ABSTRACT	III
AGRADECIMIENTOS	V
1. PREFACIO	1
1.1. Origen del trabajo y antecedentes	1
1.1.1. Nexe Fundació	2
1.1.2. Pluridiscapacidad	2
1.1.3. Filosofía de bajo coste	3
1.1.4. Silla Exploradora Inteligente 1.0	4
1.1.5. Trabajo de Fin de Grado	5
2. INTRODUCCIÓN	7
2.1. Objetivos del trabajo	8
2.1.1. Análisis de ingeniería inversa	8
2.1.2. Diseño e implementación de la solución	8
3. PROCESO DE INGENIERÍA INVERSA	12
3.1. Selección del vehículo	12
3.2. Funcionamiento del Hoverboard	15
3.3. Hardware del Hoverboard	16
3.3.1. Batería	16
3.3.2. Ruedas y motores	17
3.3.3. Placa con giroscopio	18
3.3.4. Placa controladora principal	19
3.4. Protocolo de comunicación	19
3.4.1. Análisis de la referencia	21
3.4.2. Primer proceso de ingeniería inversa	24
3.4.3. Segundo proceso de ingeniería inversa	36
3.5. Realimentación de velocidad	44
3.5.1. Ratón óptico como encoder	44
3.5.2. Sensores Hall	47
4. DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN	62

4.1. Estructura mecánica	63
4.1.1. Primer montaje.....	63
4.1.2. Segundo montaje.....	66
4.2. Etapa de potencia	68
4.2.1. Circuito de potencia	68
4.3. Sistema de control	69
4.3.1. Control con Arduino.....	69
4.3.2. Control definitivo con Arduino y Raspberry	74
5. CONCLUSIONES Y TRABAJO FUTURO	78
PRESUPUESTO Y ANÁLISIS ECONÓMICO	83
WEBGRAFÍA	86
ANEXO A	89
A1. Primer proceso de ingeniería inversa.....	89
A1.1 <i>Modificación del código de la referencia</i>	89
A2. Segundo proceso de ingeniería inversa	91
A2.1 <i>Wheelchair_sniffer.h</i>	91
A2.2 <i>Wheelchair_sniffer.cpp</i>	92
A2.3 <i>Wheelchair Motor sniffer</i>	96
A2.4 <i>Wheelchair Response Sniffer</i>	98
A2.5 <i>Gráficos MATLAB</i>	99
A2.6 <i>Control aceleración Hoverboard</i>	103
A2.7 <i>mouse_encoder.h</i>	106
A2.8 <i>mouse_encoder.cpp</i>	107
A2.9 <i>mouse_encoder_test.cpp</i>	110
A2.10 <i>Lectura de la magnitud de la velocidad</i>	111
A2.11 <i>Lectura de la dirección de la velocidad</i>	113
A2.12 <i>Control de velocidad Hoverboard</i>	115
A3. Implementación final.....	120
A3.1 <i>hoverboard_comm.ino</i>	120
A3.2 <i>hoverboard_driver_masteri2c.h</i>	127
A3.3 <i>hoverboard_driver_masteri2c.cpp</i>	129
A3.4 <i>hoverboard_driver_masteri2c_test_cpp</i>	134

1. Prefacio

1.1. Origen del trabajo y antecedentes

Este proyecto surge de la necesidad de mejorar la calidad y las condiciones de vida de las personas con avanzadas discapacidades físicas y que no disponen de los suficientes recursos económicos para poder adquirir una silla de ruedas eléctrica convencional.

Por ello, se considera que se debe hacer mención especial a la Fundación Nexe y, en concreto, a su primer prototipo de plataforma de ruedas inteligente adaptable a varios sistemas de sedestación, mostrado en la figura 1.1:

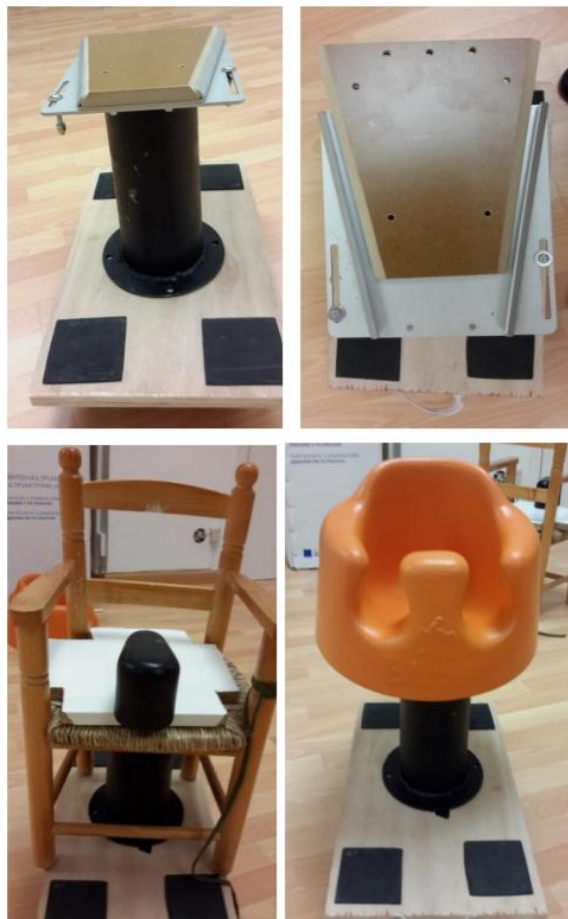


Figura 1.1.- Diferentes tipos de soportes y asientos colocados sobre la plataforma móvil inteligente de Nexe Fundació. [2]

1.1.1. Nexe Fundació

Nexe Fundació [3] es una entidad privada especializada en la atención a niños con pluridiscapacidad y a sus familias, que ofrece recursos globales e individualizados a las necesidades educativas, sanitarias y psicosociales del niño desde el momento de su nacimiento y de apoyo a la familia.

Nexe se constituyó como Fundación en 1991, en el marco de uno de los primeros centros de atención precoz de Cataluña en 1981, para potenciar nuevas intervenciones y actividades dirigidas a la atención global e interdisciplinaria de los niños con graves trastornos en el desarrollo y a sus familias. A lo largo de más de 30 años de trayectoria, aportando soluciones a las necesidades de los niños, de las familias y de los profesionales, Nexe ha creado una red de servicios innovadores y pioneros reconocidos a escala estatal y europea.



Figura 1.2.- Logotipo de Nexe Fundació [3].

1.1.2. Pluridiscapacidad

Se ha considerado importante hacer una breve explicación sobre la pluridiscapacidad, de modo que el lector tenga una visión más clara de las motivaciones y las necesidades existentes al realizar este proyecto.

La pluridiscapacidad es una discapacidad grave de origen neurológico que se manifiesta a través de un retraso importante del desarrollo psicomotriz, acompañado frecuentemente de déficits sensoriales (visión, audición...), crisis epilépticas y problemas de salud: digestivos, respiratorios, cutáneos, dentales, cardiovasculares... [4]

El perfil de un niño con pluridiscapacidad es muy diverso y, consecuentemente, sus necesidades también lo son, pero en todos los casos se da una restricción importante de las capacidades de percepción, comunicación y autonomía, que se traduce en una gran dependencia del niño. Por eso hace falta una intervención global en los ámbitos médico, sanitario, psicopedagógico y social.

Consecuentemente, se puede decir que la motivación principal para realizar este proyecto fue poder ayudar a estos niños que más lo necesitan, y todo gracias a los conocimientos en ingeniería que he adquirido a lo largo de mi carrera.

1.1.3. Filosofía de bajo coste

Seguidamente se introduce, con un fragmento [5] extraído de la página web de Nexe Fundació, el término bajo coste o filosofía de bajo coste, eje principal del proyecto:

"Con bajo coste nos referimos a los recursos que uno puede construir, o bien que puede conseguir sin coste, sea porque se bajan de la red o porque se obtienen de forma gratuita para el usuario. Con una perspectiva más amplia incluiríamos también aquellos recursos de mercado general que pueden utilizarse como productos de apoyo.

El principal elemento del bajo coste es el conocimiento. Sin conocimiento es imposible utilizar las soluciones del bajo coste. El bajo coste, muy cercano al "hazlo tú mismo", requiere conocimientos que proporcionan autonomía y autosuficiencia. Los productos de bajo coste siempre dan respuesta a una necesidad concreta, aunque no se haya realizado un análisis profundo de la situación o de las posibles soluciones. Para los usuarios y familiares en muchas ocasiones esta solución artesanal es la única posible de forma temporal o definitiva."

Dado a las circunstancias de personas con necesidades especiales a causa de su discapacidad, y teniendo en cuenta la filosofía bajo coste, Nexe Fundació creó un catálogo de proyectos el cual ha dado respuestas a necesidades específicas, donde cada idea está adaptada y documentada para que, siguiendo los pasos indicados, se pueda reproducir el modelo de la forma más sencilla posible.

En la figura 1.3 se muestra uno de estos proyectos, en concreto, un chupete con pulsador de bajo coste para ser accionado con la boca, con el cual se pueden activar juguetes o dispositivos electrónicos:

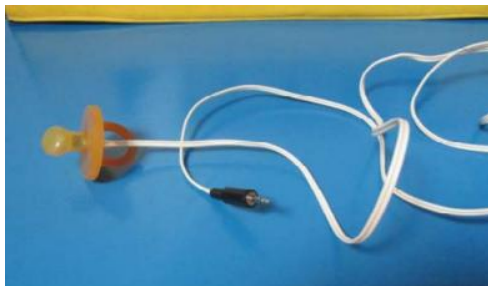


Figura 1.3.- Chupete pulsador de bajo coste [6].

1.1.4. Silla Exploradora Inteligente 1.0

Este Trabajo de Fin de Grado está directamente relacionado con una de estas ideas de bajo coste: la Silla Exploradora Inteligente 1.0. [2] [7]

La Silla Exploradora Inteligente (versión 1.0) fue creada para que niños y niñas gravemente afectados puedan experimentar el movimiento de forma autónoma y segura por entornos de interior controlados (casa, colegio...).

Está compuesta por una plataforma con ruedas y por un sistema de sedestación adaptado a las características del niño o niña que lo utilice, como el mostrado en la figura 1.1. La plataforma se pone en movimiento gracias a unos motores que se activan accionando un conmutador, y tiene estratégicamente colocados unos sensores para detectar obstáculos y cambiar la trayectoria del recorrido cuando sea necesario. Se definiría como una silla de aprendizaje que permite:

- Descubrir la relación causa-efecto de manera vivencial.
- Experimentar el movimiento de forma autónoma.
- Desplazarse por un entorno controlado de manera automática y segura.

Esta versión está pensada para pediatría, pero se puede replicar el modelo para adolescentes, adultos y personas mayores, simplemente cambiando la capacidad de los motores y de las baterías, y poniendo unas ruedas que soporten el peso de la persona que la va a utilizar.

También está diseñada para ser accionada desde una Tablet o dispositivo móvil, pero puede adaptarse para ser utilizada incluso a través de la voz o sistemas de barrido. La Silla Exploradora Inteligente (versión 1.0) cumple los requisitos de la filosofía de bajo coste, los cuales son:

- Bajo precio (si se compara con una silla de ruedas eléctrica de ortopedia).
- Accesible a personas con discapacidad motriz y cognitiva.
- Replicable porque utiliza tecnología libre.
- Universal, pues con una única plataforma, pueden ponerse diferentes sistemas de sedestación en función de las características de cada niño.

La figura 1.4 muestra la Silla Exploradora Inteligente 1.0, donde el pulsador de color amarillo activa el movimiento de la silla:



Figura 1.4.- Silla Exploradora Inteligente 1.0 creada en Nexe Fundació. [7]

1.1.5. Trabajo de Fin de Grado

Nexe Fundació contactó con el Institut de Robòtica i Informàtica Industrial CSIC-UPC para hacer una mejora del primer prototipo mencionado en el apartado anterior.

El principal objetivo de esta mejora era poder implementar un sistema de navegación a la silla de ruedas vista en la figura 1.4, para que pudiese moverse autónomamente, robustamente, y de forma segura, por entornos interiores como puede ser un piso o un hospital, aparte de mejorar las prestaciones de este primer prototipo, como podría ser aumentar la carga máxima que puede soportar la silla (diseñarla para todos los públicos) y desarrollar un control de velocidad mejorado.

Posteriormente, el IRI ofertó la posibilidad de escoger este proyecto como Trabajo de Fin de Grado y, finalmente, contacté con ellos para ofrecirme como proyectista de este.



Figura 1.5.- Logo del Institut de Robòtica i Informàtica Industrial.



2. Introducción

En el presente Trabajo de Fin de Grado se ha desarrollado una silla de ruedas eléctrica de bajo coste con control de velocidad por joystick.

Cabe aclarar, que este trabajo es una parte del proyecto del Institut de Robòtica i Informàtica Industrial "*Diseño y construcción de una silla exploradora inteligente de bajo coste*" [1], dado que el objetivo final que engloba este proyecto es desarrollar una silla de ruedas autónoma de bajo coste, basándose en las mejoras requeridas de la Silla Exploradora Inteligente 1.0 que se creó en Nexe Fundació. La figura 2.1 pretende mostrar la finalidad del proyecto descrito en esta memoria:

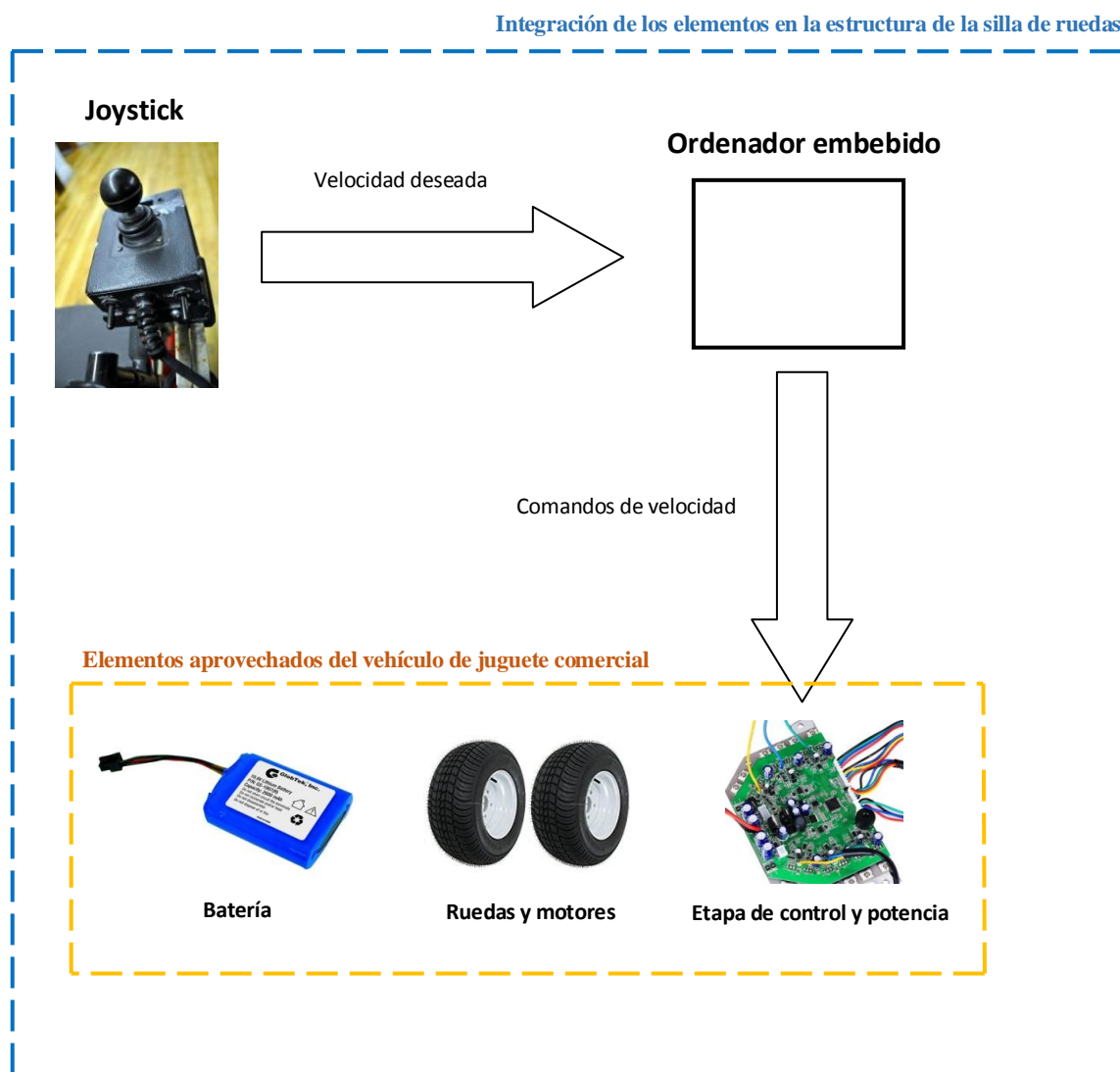


Figura 2.1.- Diagrama de alto nivel de la finalidad del proyecto.

2.1. Objetivos del trabajo

El objetivo principal de este proyecto es responder a las necesidades que se plantean desde el proyecto de Nexe Fundació, es decir, crear una plataforma móvil con prestaciones a nivel mecánico y electrónico optimizadas en comparación a la Silla Exploradora Inteligente 1.0, además de mejorar el control de movimiento de la plataforma. Para ello, se añadirá un joystick con el cual se realizarán las pruebas necesarias del control de movimiento de la silla.

Paralelamente, siempre se tendrá en cuenta optimizar al máximo los costes del proyecto, con el objetivo de seguir la filosofía de bajo coste. De esta manera, cualquier persona será capaz de replicar esta misma plataforma sin tener que adquirir una silla de ruedas eléctrica a precio de mercado.

Para poder llegar al objetivo final, se han marcado una serie de objetivos específicos, los cuales se agrupan en dos actividades.

2.1.1. Análisis de ingeniería inversa

El objetivo de esta actividad es realizar técnicas de ingeniería inversa en un vehículo de juguete comercial y de precio asequible, del cual se pueda aprovechar gran parte del material (motores, ruedas, batería, cargador, controladora, etc), para así poder cumplir la filosofía de bajo coste. También se debe tener en cuenta que el dispositivo escogido sea fácil de conseguir, dado que eso facilitará la reproducibilidad y así cualquier persona podrá disponer de la plataforma móvil.

2.1.2. Diseño e implementación de la solución

El objetivo de esta actividad es diseñar una estructura mecánica de bajo coste para la plataforma móvil que permita moverse sin problemas en entornos interiores, principalmente superficies llanas o rampas de poca pendiente, para más adelante poder implementar un control de velocidad a la silla de ruedas. Para ello, se realizarán las siguientes sub actividades:

- Diseñar una estructura de bajo coste, pero suficientemente robusta, que pueda soportar el peso de una persona adulta, además de poder contener todos los elementos necesarios para su control, como pueden ser la batería, los motores, la controladora, etc.

- Integrar los elementos eléctricos y electrónicos del vehículo de juguete comercial escogido a la plataforma mecánica diseñada.
- Añadir un control de velocidad a bajo nivel que permita mover la silla de ruedas diseñada mediante un joystick, con el objetivo de confirmar el correcto funcionamiento de la plataforma.
- Una vez se verifique el funcionamiento del control de velocidad a bajo nivel, realizar un control a más alto nivel utilizando un ordenador embebido, como puede ser BeagleBoard o Raspberry Pi, que permitirá en un futuro utilizar ROS (Robot Operating System) [8], donde se implementarán los algoritmos necesarios de SLAM [9] para implementar la navegación autónoma de la silla de ruedas [10].

3. Proceso de ingeniería inversa

3.1. Selección del vehículo

Normalmente, para escoger qué elementos generaran el movimiento de la silla de ruedas (motores, batería, etc) bastaría con buscar proveedores de dichos componentes y comprarlos, buscando el equilibrio entre calidad y precio.

Si uno de los objetivos de este proyecto es seguir la filosofía de bajo coste y, por ende, optimizar al máximo los costes y la complejidad a la hora de diseñar la silla de ruedas, la pregunta que puede surgir es si realmente es posible encontrar una mejor solución que comprar los componentes necesarios para diseñar la silla de ruedas (batería, motores, controlador, etc) por separado.

Para tener una primera idea de cuánto se está optimizando realmente el precio del prototipo, se hizo una búsqueda del coste habitual de una silla de ruedas eléctrica convencional. Este precio varía, desde 950 € aproximadamente, las más simples, hasta 3000€ las de más calidad.

También hay que tener en cuenta el precio total de la Silla Exploradora Inteligente 1.0 de bajo coste que crearon en Nexe Fundació, el cual gira en torno a los 300€ [2]. Como se puede observar, se disminuyeron los costes considerablemente si se compara con el precio una silla de ruedas eléctrica convencional.

Para hacer un análisis más concreto, los costes que tuvieron los componentes principales que dan movimiento a la silla, según la memoria de su proyecto [2], fueron de 170 €, y desglosado:






- Dos ruedas motrices: 40 €
- Dos ruedas directrices (ruedas locas): 30 €
- Dos motores de 6 V y batería de 12 V: 40 €
- Etapa de control (Controladora Shield para motores DC): 60 €

Seguidamente, se realizó un estudio comparativo entre el precio de los componentes mostrados anteriormente, con las dos opciones a seguir para la selección de componentes de la silla de ruedas de este proyecto, las cuales son:

- Comprar los componentes y elementos por separado, como se hizo en el diseño de la Silla Exploradora Inteligente 1.0 de Nexe Fundació.
- Aprovechar un vehículo de juguete del mercado, y sacarles el máximo provecho posible a sus componentes.

La tabla 3.1 muestra las diferencias existentes entre todas las opciones estudiadas (todos los valores mostrados son aproximados, ya que el mercado es muy extenso y existen multitud de modelos):

Tabla 3.1. - Tabla comparativa entre distintos vehículos de juguete.

	Vehículo	Ruedas	Motores	Batería	Carga soportada máx.	Precio
	E-Bike	Demasiado grandes Ruedas de 26" o 29"	1 motor 350 W - 500 W Necesitaríamos otro motor	Batería 30 V 10 Ah	100 kg - 120 kg	1500 € - 5000 €
	Patinete eléctrico	Se pueden aprovechar Ruedas de 10"	1 motor de 24 V 280 W Necesitaríamos otro motor	Batería de 24 V 10 Ah	100 kg - 110 kg	350 € - 800 €
	Hoverboard	Se pueden aprovechar Ruedas de 6,5" o 10"	Se pueden aprovechar 2 motores de 350 W	Batería de 36 V 4,4 Ah	100 kg - 120 kg	200 € - 280 €
	Coche de juguete	Se puede aprovechar Ruedas de 6,5" o 10"	1 motor de 6 V Motor muy pequeño Necesitaríamos otro motor	Batería de 6 V 7 Ah	20 kg	100 € - 250 €
	Partes por separado	Ruedas de 10" 50 €	2 motores de 24 V 250 W 80 €	Batería 36 V 4,4 Ah 100 €	-	230 €

En la tabla 3.1 se analizan las características de 4 vehículos de juguete diferentes, y en la última fila aparece lo que costaría comprar cada componente requerido para construir la silla de ruedas por separado. Los vehículos corresponden a una bicicleta eléctrica (*E-bike* en inglés), un patinete eléctrico, una tabla de dos ruedas autoequilibrada (*Hoverboard* en inglés) y un coche eléctrico de juguete eléctrico para niños.

Las casillas de color rojo son las características de los componentes de cada vehículo que no se pueden aprovechar, debido a su limitación en las prestaciones, tamaño o precio, mientras que en verde son los elementos que sí se consideran aprovechables.

Observando el precio de cada uno de los vehículos, se puede prescindir inmediatamente de la bicicleta eléctrica, ya que tiene un coste bastante elevado y por lo tanto no es adecuado para este proyecto de bajo coste.

Por otra parte, el coche de juguete tiene un coste muy bajo. Aun así, se puede ver que solo tiene un motor de 6 V, lo que limitaría el movimiento de la plataforma resultante, ya que estos motores no podrían mover el peso de una persona adulta, y no se podría cumplir el objetivo de crear una silla que pueda ser utilizada por todos los públicos. Por lo tanto, tampoco se considera adecuada la adquisición de este vehículo de juguete para el proyecto.

Finalmente, los posibles vehículos de juguete candidatos a ser reutilizados son el patinete eléctrico (scooter) y el Hoverboard. Tienen un coste de compra bastante idóneo (aunque los modelos más completos del patinete son caros), y tanto las ruedas como la batería se pueden reutilizar en ambos casos. Sin embargo, el patinete eléctrico posee un sólo motor y, como en el caso de la bicicleta eléctrica o el cochecito eléctrico de juguete, también limitaría el movimiento de la silla.

Gracias al análisis realizado anteriormente en base a las prestaciones, precio y limitaciones de cada vehículo, se concluyó que el Hoverboard era el vehículo de juguete más conveniente para aprovechar sus componentes.

Por otra parte, si los componentes se hubiesen comprado por separado, tampoco hubiese sido una mala opción, si se tiene en cuenta el precio (230 € aproximadamente). Lo que realmente materializó la compra del Hoverboard fue el hallazgo de una publicación en un blog de hacking y electrónica, llamado "*Hackaday*" [11].

En este blog se explica el análisis de ingeniería inversa realizado para descubrir el protocolo de comunicación que utiliza la electrónica del vehículo. Además, consigue controlar, con un software externo, una de las ruedas del Hoverboard, aunque no logra controlar las dos ruedas al mismo tiempo.

Aun así, se creyó muy conveniente seguir adelante con la opción de compra del Hoverboard, ya que a partir de este proceso de ingeniería inversa se podría aprovechar también la etapa de control del vehículo (no aparece en los costes de la tabla 3.1), y así reducir todavía más los costes de material del proyecto. La figura 3.1 muestra el Hoverboard adquirido para el proyecto.



Figura 3.1.- Hoverboard adquirido para el proyecto. [12]

3.2. Funcionamiento del Hoverboard

El funcionamiento del Hoverboard es sencillo. A medida que la persona se inclina hacia adelante, el Hoverboard avanza en ese sentido y, cuanto más inclinado está el cuerpo de la persona con respecto a su eje de equilibrio, más rápido se mueve. Lo mismo ocurre si vas hacia detrás. [13].

La figura 3.2 muestra este principio de funcionamiento, donde las flechas indican el sentido del movimiento, y la línea discontinua corresponde al eje de equilibrio de la persona.

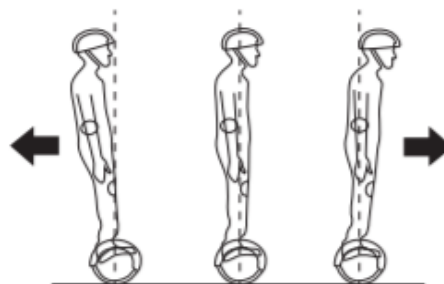


Figura 3.2.- Según la persona se mueve con respecto a su eje de equilibrio, el Hoverboard avanza en una dirección u otra. [13]

Por otra parte, para poder girar basta con mover los pies hacia delante o hacia detrás encima de la plataforma, para así cambiar el centro de masa de tu cuerpo y hacer que el Hoverboard gire, como indica la figura 3.3:



Figura 3.3.- Funcionamiento de giro del Hoverboard. [13]

3.3. Hardware del Hoverboard

Para comprobar el hardware del Hoverboard y verificar si realmente estos elementos electrónicos eran aprovechables para el diseño de la silla de ruedas, se procedió a abrir su chasis, como se observa en la figura 3.4:

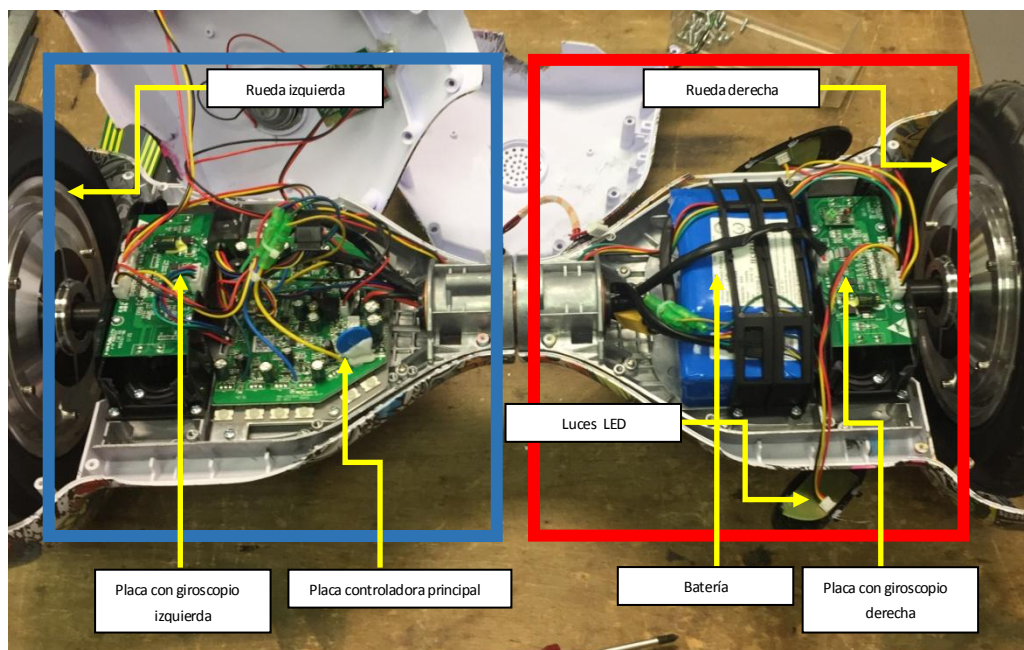


Figura 3.4.- Hoverboard con el chasis inferior abierto, con sus componentes visibles.

En la figura 3.4 se pueden diferenciar dos partes: la parte izquierda (recuadro azul), donde se encuentra la rueda izquierda, la placa con giroscopio izquierda y la placa controladora principal, y la parte derecha (recuadro rojo), donde se encuentra la rueda derecha, la batería y la placa con giroscopio derecha.

En las siguientes secciones se explica en qué consisten estos componentes más detalladamente:

3.3.1. Batería

Se trata de una batería de ion de litio, de 36 Vdc y 4,4 Ah, que alimenta la placa controladora principal del Hoverboard.

Se quiere aprovechar esta batería ya que podría alimentar a todo el sistema de la silla de ruedas, aportando una autonomía decente, en concreto, de 15 km a 20 km aproximadamente, dependiendo del peso y de la velocidad en que se utiliza el Hoverboard.[13]

En la figura 3.5 se muestra la batería del Hoverboard, y en la figura 3.6 sus especificaciones:



Figura 3.5.- Batería de ion de litio del Hoverboard.

NAME	PARAMETERS
Battery	Lithium-ion battery
Charging Time	2-3 Hours
Voltage	36V
Initial Volume	4.4 AH
Working Temperature	-15 ~ 50°C
Charging Temperature	0 ~ 40 °C
Relative Humidity of Storage	5% ~ 95%

Figura 3.6.- Especificaciones de la batería del Hoverboard. [13]

3.3.2. Ruedas y motores

Son dos ruedas de 10 " de diámetro (25,4 cm). Cada una de ellas contiene en su interior un motor DC sin escobillas con alimentación trifásica, y tres sensores Hall que informan a la controladora de la posición del motor.

Se quiere aprovechar ambas ruedas, ya que tanto el tamaño del neumático como la potencia de los motores (350 W cada uno) [13] , se considera adecuado para las especificaciones de la silla de ruedas del proyecto y, en consecuencia, podrían soportar perfectamente la carga de un humano adulto.

Además, si existe la posibilidad de leer los sensores Hall que lleva incorporados, también se podría obtener una realimentación de velocidad de las ruedas, necesaria para el control de velocidad con joystick de la silla. Con esta solución no haría falta adquirir ningún sensor extra para realizar esta lectura, como podría ser un encoder.

En la figura 3.7 se observa una imagen del interior de una de las ruedas:

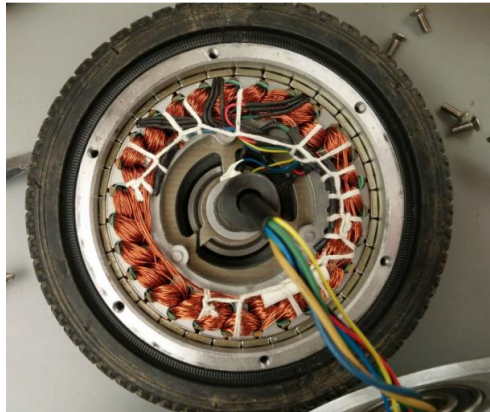


Figura 3.7.- Aspecto interior de la rueda del Hoverboard, con el bobinado del motor visible.

3.3.3. Placa con giroscopio

Estas placas son las encargadas de enviar la información sobre la inclinación del usuario a la controladora principal, gracias a los giroscopios que tiene incorporados. También tiene conectadas varias luces LED, pero estas solo tienen el objetivo de embellecer el Hoverboard.

Como se observa en la figura 3.8, disponen también de dos interruptores ópticos los cuáles, cuando son tapados a causa del peso del usuario, informan a la controladora principal de que alguien está utilizando el Hoverboard. Estos interruptores se utilizan como medida de seguridad para que, al encender el dispositivo, el Hoverboard no avance si no hay nadie montado previamente.

Estas placas no se pretenden reutilizar ya que no contienen ningún elemento electrónico que sea de interés para la silla de ruedas.

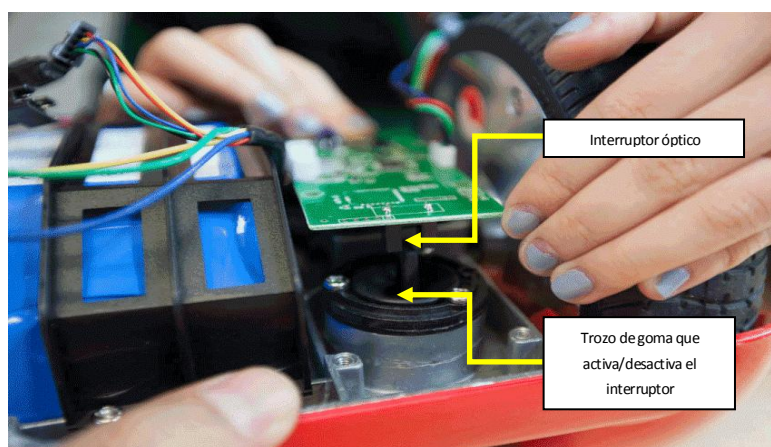


Figura 3.8.- La placa con giroscopio (en verde) y su interruptor óptico. [14]

3.3.4. Placa controladora principal

En la placa controladora principal, mostrada en la figura 3.9, con sus elementos y cables de conexión principales, se encuentra la etapa de potencia y de control del Hoverboard. El elemento más importante de esta placa es un microcontrolador ARM, el cual se encarga de procesar la información de las placas con giroscopios, el estado de la batería y de controlar las ruedas, leyendo la velocidad de estas gracias a los sensores Hall que hay en los motores.

En este caso, se quiere reutilizar esta placa controladora principal ya que posee la etapa de potencia que alimentaría a los motores y, si además se consigue reproducir la comunicación de las placas con giroscopio con un software propio, también se podría aprovechar la etapa de control.

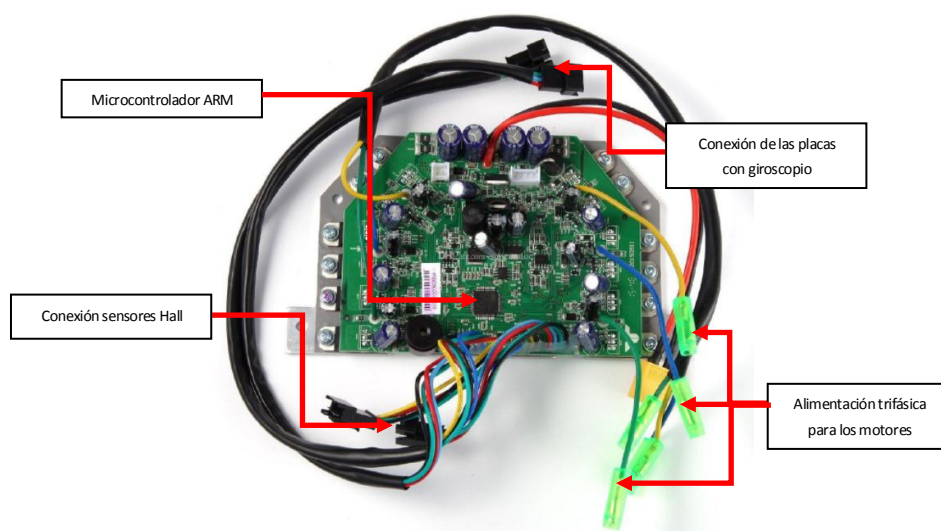


Figura 3.9.- Placa controladora principal del Hoverboard, con sus respectivos cables de conexión a placas con giroscopio y sensores Hall del motor.

3.4. Protocolo de comunicación

Una vez conocida la electrónica del Hoverboard y después de haber verificado qué componentes se pueden reutilizar para el diseño de la silla de ruedas eléctrica, el siguiente objetivo del proyecto fue descifrar el protocolo de comunicación del Hoverboard, siguiendo los pasos del análisis de ingeniería inversa realizado en la referencia [15].

Como se ha comentado en el apartado 3.3, las placas con giroscopio son las encargadas de leer la inclinación del usuario gracias a los giroscopios, además de detectar la presencia de la persona gracias a los interruptores ópticos. Seguidamente, esta información se envía a la placa

controladora principal para que procese estos datos y envíe el comando de velocidad correspondiente a las ruedas en función de esa inclinación, como se muestra en la figura 3.10:

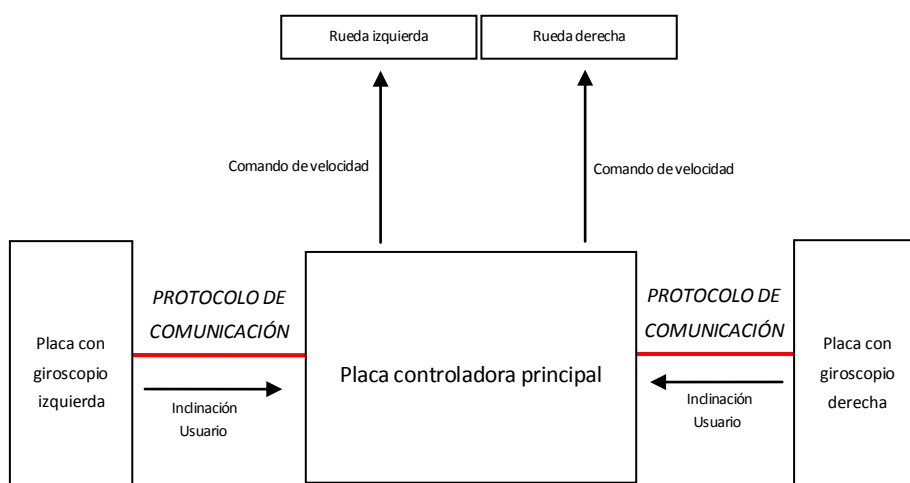


Figura 3.10.- Diagrama de alto nivel de la comunicación entre las placas del Hoverboard.

Por lo tanto, si se quiere controlar la velocidad de las ruedas del Hoverboard sin las placas con giroscopio de serie, hay que simular la inclinación que enviarían estas cuando el usuario se balancea para poder adquirir velocidad con el vehículo. En la figura 3.11 se muestra el diagrama de alto nivel que se desea realizar:

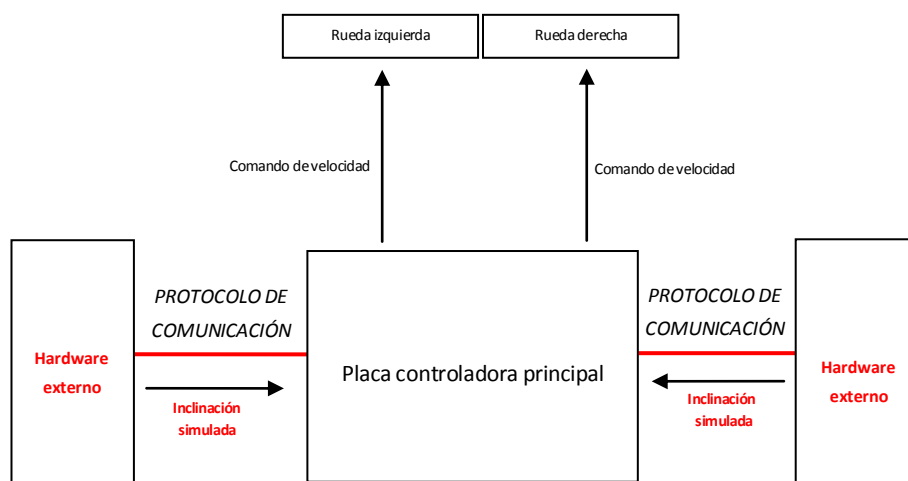


Figura 3.11.- Diagrama de alto nivel de la comunicación que se desea realizar: A través de un software y hardware externo se enviará la información necesaria a la controladora del Hoverboard.

En definitiva, en el apartado 3.4 se pretende mostrar el proceso de ingeniería inversa llevado a cabo para conocer como se comunican las placas con giroscopio con la controladora principal. Principalmente, qué protocolo de comunicación utilizan y qué estructura tiene la trama de

información enviada entre estos dispositivos, con el objetivo final de realizar un control de velocidad propio que pueda ser implementado en la silla de ruedas eléctrica del proyecto.

Cabe señalar, otra vez, que para realizar este proceso de ingeniería inversa se ha tomado como referencia el blog de electrónica [15] mencionado anteriormente, donde se ha conseguido descifrar el protocolo de comunicación que utiliza internamente, además de controlar una de las ruedas del Hoverboard. Por lo tanto, se introducirá primeramente un análisis de esta referencia en cuestión.

3.4.1. Análisis de la referencia

En este apartado se exponen las conclusiones extraídas del proceso de ingeniería inversa de la referencia.

Se concluye que el protocolo de comunicación que utilizan las placas con giroscopios y la placa controladora principal para comunicarse entre ellas es UART (Universal Asynchronous Receiver-Transmitter) [16].

Concretamente, la comunicación del Hoverboard (según la referencia [15]) tiene las siguientes características:

- La velocidad de transmisión es de 26315 baudios.
- 1 bit de START.
- 9 bits de datos.
- No hay paridad.
- 1 bit de STOP.
- La información se envía con el bit menos significativo primero (LSB-first).

La estructura de la trama enviada desde las placas con giroscopio a la placa controladora principal se compone de 6 números de 9 bits, donde cada número tiene una función específica, como muestra la figura 3.12:



Figura 3.12- Estructura de los datos enviados por la placa con giroscopio a la controladora del Hoverboard.

- 256: Indica el inicio de la trama de datos. Este valor siempre es fijo (256).
- X: Byte bajo de la inclinación del giroscopio.

- Y: Byte alto de la inclinación del giroscopio.
- $X+256 \cdot Y$: Palabra proporcional a la inclinación del giroscopio.
- 170/85: Indica si hay alguien montado (85) o no (170). (Sensado por los interruptores ópticos).

En general, la información que las placas con giroscopio envían a la controladora principal es la mostrada en la figura 3.13:

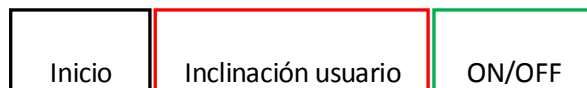


Figura 3.13- Estructura simplificada de los datos enviados por la placa con giroscopio a la controladora principal.

3.4.1.1. Implementación del control

Para esta primera implementación del control se utilizó un ordenador de sobremesa, una placa Arduino Nano y la controladora principal del Hoverboard, tal y como se describe en la referencia. En la figura 3.14 se representa la implementación realizada:

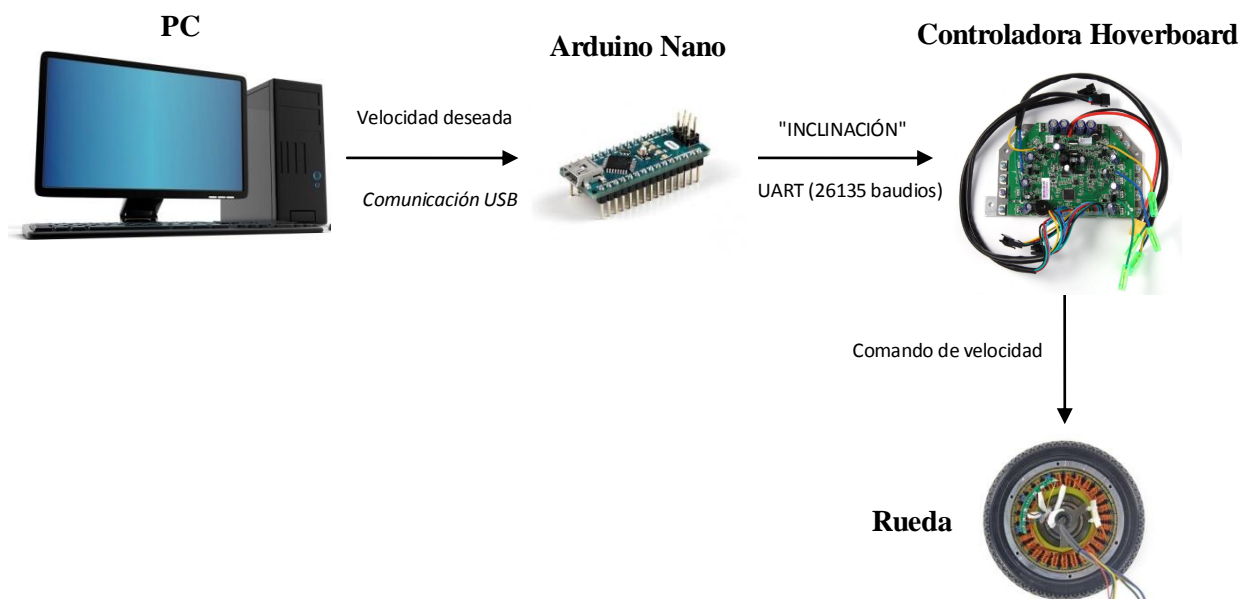


Figura 3.14.- Implementación del control realizado en la referencia

Desde el ordenador se envía por comunicación USB los comandos de inclinación deseados a la placa Arduino, la cual se comunica con la controladora principal por comunicación UART, informándole de la inclinación del usuario (en este caso, esta inclinación es simulada por software ya que ningún usuario está utilizando el Hoverboard y por lo tanto no hay placa con giroscopios).

Finalmente, la controladora principal envía los comandos de velocidad necesarios para que la rueda se mueva a una velocidad concreta, proporcional a la inclinación simulada.

La conexión entre la controladora principal del Hoverboard y la placa Arduino Nano es la representada en la figura 3.15:

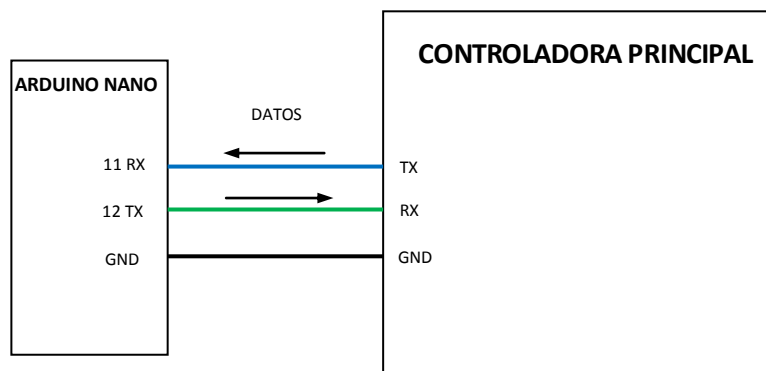


Figura 3.15.- Diagrama de la comunicación entre Arduino Nano y controladora principal.

En ningún momento se explica en la referencia qué tipo de información se envía en sentido contrario (de la controladora principal del Hoverboard a la placa con giroscopio), por lo que para realizar los primeros experimentos se obvió esa información. Aun así, se necesitan conectar ambos hilos para que la comunicación UART funcione correctamente.

Para poder enviar las tramas de 9 bits se utiliza la librería de Arduino SoftwareSerial [17]. Esta librería ha sido desarrollada originalmente para soportar una comunicación serial de 8 bits entre dispositivos, por lo que en la referencia se tuvo que modificar para trabajar con tramas de 9 bits [18].

3.4.1.2. Resultados

A pesar de haber seguido los pasos del análisis de ingeniería inversa realizado en la referencia, no se consiguió ninguna respuesta por parte de las ruedas del Hoverboard utilizando el código proporcionado por esta. Se barajaron diversas hipótesis respecto a por qué el software proporcionado [19] no funcionaba para el Hoverboard adquirido en el proyecto:

- El protocolo de comunicación no es UART.
- La velocidad de transmisión de la comunicación serial es distinta.
- La estructura de la información enviada entre las placas con giroscopios y la controladora principal (figura 3.12) es distinta.

Para comprobar las hipótesis anteriores, se realizó un análisis de ingeniería inversa con el propio Hoverboard del proyecto, explicado en el siguiente apartado 3.4.2.

3.4.2. Primer proceso de ingeniería inversa

Para comprobar que el protocolo de comunicación entre placas era UART (tal y como se expone en la referencia), se capturaron los datos de la comunicación entre la placa con giroscopio y la placa controladora principal del Hoverboard trabajando en vacío, sin peso.

Concretamente, se capturó la información pinchando con el osciloscopio en el conector JST de 4 pines del bus de comunicación entre la placa con giroscopio y la placa controladora principal, tal y como se representa en las figuras 3.16 y 3.17:

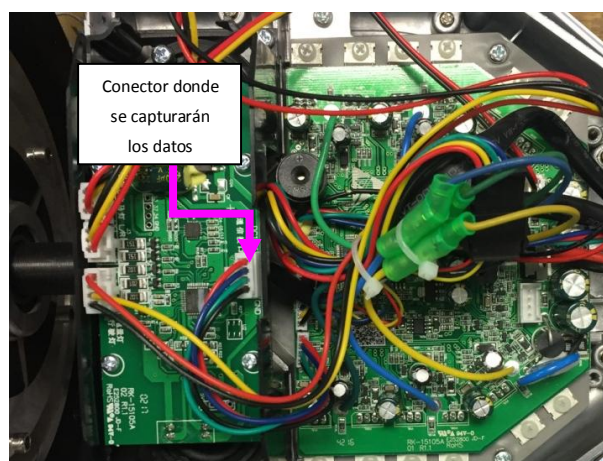


Figura 3.16.- La flecha lila señala el conector JST donde se realizará la captura de los datos de la comunicación entre placas.

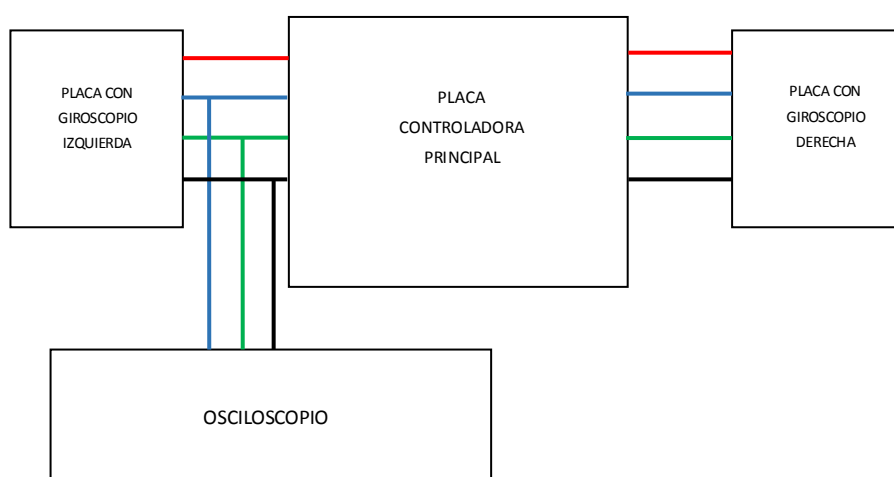


Figura 3.17.- Representación del experimento para la captura de datos de la comunicación entre placas.

Sólo son de interés las señales de los cables interiores (azul y verde), las que corresponden supuestamente a los hilos de comunicación UART, ya que los cables externos corresponden a un voltaje de alimentación (cable rojo) y masa (cable negro).

La figura 3.18 muestra una de las señales capturadas con el osciloscopio, en concreto, la que corresponde a la información que envía la placa con giroscopio a la controladora:

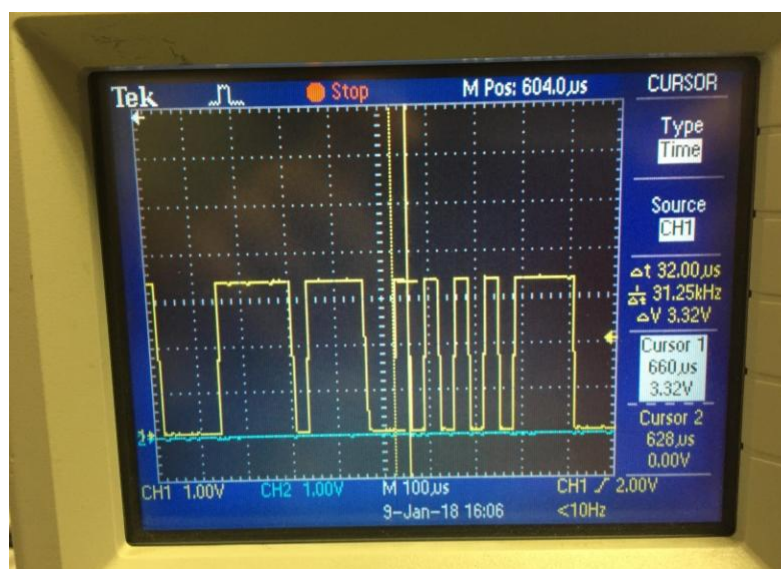


Figura 3.18. Señal capturada en el osciloscopio correspondiente a las tramas enviadas desde la placas con giroscopio a la placa controladora principal del Hoverboard.

Se observa que la señal tiene un nivel lógico de 3,3 V y que toda la información se envía con un tiempo de bit múltiple de 32 us. Por lo tanto, eso descarta la posibilidad de que la información se codifique como PWM (Pulse Width Modulation).

Por otra parte, si fuese un protocolo I2C (Inter-Integrated Circuit), una de las dos señales obtenidas tendría que corresponder a una señal de reloj en el momento que la otra señal envía información, y en ningún caso aparece una señal como esa en el osciloscopio. En consecuencia, si no hay una señal de reloj se tratará de un protocolo de comunicación asíncrono, por lo que se elimina la posibilidad de que se trate de un protocolo I2C.

Por descarte, el único candidato posible como protocolo de comunicación del Hoverboard es UART, tal y como indicaba la referencia original.

El tiempo de bit de 32 us corresponde a una velocidad de transmisión de 31250 baudios. En consecuencia, la velocidad de transmisión que se utilizó anteriormente en el código de la referencia, la cual era de 26315 baudios, no era la correcta.

Además, analizando el contenido de la trama de datos enviada por el giroscopio, se descubren dos bytes más, los cuales no existían en la trama de datos original comentada en la referencia. Estos dos nuevos bytes de información, con un valor igual a 88, se pueden observar en la trama capturada de la figura 3.19:

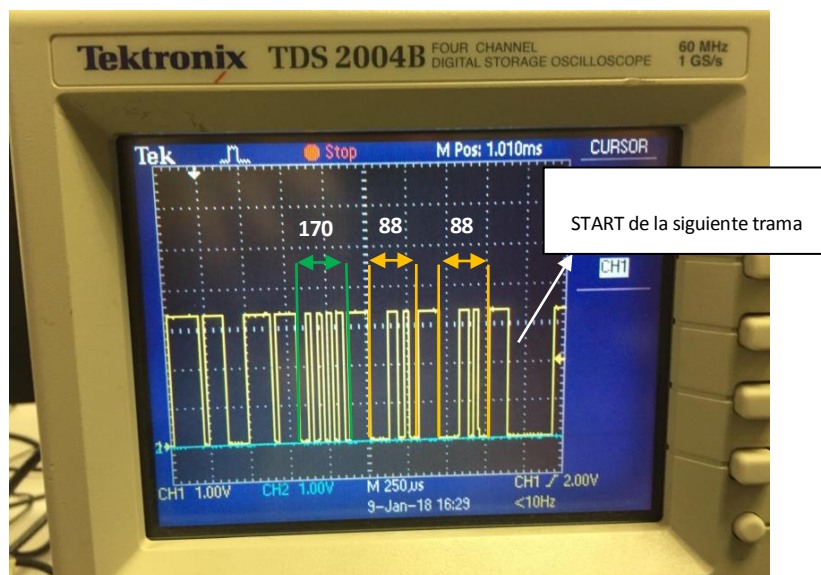


Figura 3.19. Los dos nuevos bytes encontrados dentro de la trama enviada por la placa con giroscopio a la controladora principal del Hoverboard

Después de varios experimentos con las placas con giroscopio, se concluyó que estos dos valores eran proporcionales a la aceleración del Hoverboard respecto a su eje central ya que, si se realizaba un desplazamiento lineal con la placa, el valor de estos bytes no cambiaba. En cambio, si se realizaban movimientos de giro, el valor se veía modificado, y cuanto más rápido se giraba, más se modificaba. La figura 3.20 muestra este comportamiento de los nuevos bytes encontrados:

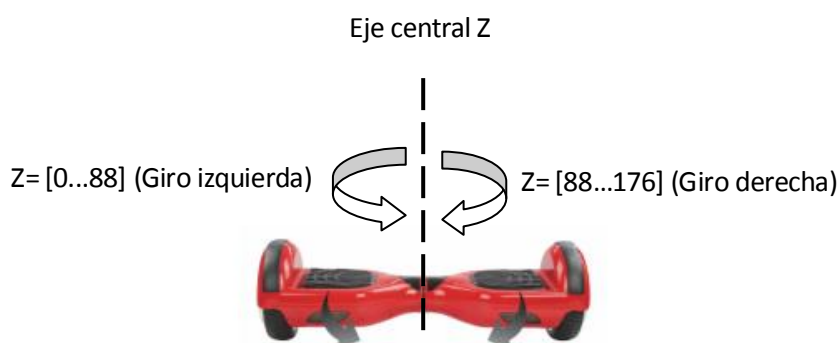


Figura 3.20.- Comportamiento de los bytes de aceleración según el lado hacia donde se gira, con respecto al eje central del Hoverboard.

A diferencia de lo que se expone en la referencia, la trama de datos que envían las placas con giroscopio en el Hoverboard del proyecto consiste de 8 números de 9 bits cada uno, como se muestra en la figura 3.21:



Figura 3.21.- Estructura de los datos enviados por la placa con giroscopio.

Donde:

- 256: Indica el inicio de la trama.
- X: Corresponde al byte bajo de la inclinación.
- Y: Corresponde al byte alto de la inclinación.
- $X + 256 \cdot Y$: Es una palabra directamente proporcional a la inclinación del giroscopio.
- 170/85: Indica si hay alguien montado (85) o no (170).
- Z: Corresponde a un valor proporcional a la aceleración en el eje Z del Hoverboard.

Realmente no se descubrió porqué se repiten los bits que corresponden a la inclinación (X e Y) y aceleración angular (Z) en la trama enviada por las placas con giroscopio. La principal hipótesis es que, al ser una comunicación sin paridad y, por lo tanto, no hay detección de errores, de alguna manera el microcontrolador comprueba que la comunicación es correcta comprobando que estos valores son iguales.

Como resumen, en la tabla 3.2 se muestra una comparativa entre las características de la comunicación del modelo de Hoverboard de la referencia y el del proyecto:

Tabla 3.2.- Tabla comparativa de las características de la comunicación entre la referencia y el proyecto.

	Hoverboard referencia	Hoverboard del proyecto
Protocolo de comunicación	UART	UART
Velocidad de transmisión	26315 baudios	31250 baudios
Estructura de los datos	6 números de 9 bits	8 números de 9 bits

Finalmente, y una vez encontradas las discrepancias vistas en la tabla 3.2 en la velocidad de transmisión y la estructura de los datos enviados, se modificó el código de la referencia para adaptarlo a las necesidades de la comunicación del Hoverboard del proyecto.

Para ello, se modificó la velocidad de transmisión de la comunicación UART del código proporcionado en la referencia a 31250 baudios, y se añadieron 2 bytes más a la trama de datos que envía Arduino.

El código completo modificado se puede ver en el anexo [A1.1].

Una vez realizadas las modificaciones en el programa, las ruedas del Hoverboard ya respondían a los comandos enviados desde el ordenador, y se pudo comprobar que la comunicación UART entre dispositivos funcionaba correctamente. Se realizaron diversas pruebas para tratar de entender mejor la relación entre la trama de datos enviada desde Arduino Nano y la respuesta de las ruedas del Hoverboard.

Tal y como se ha comentado anteriormente, en la referencia [15] se ha logrado controlar una sola rueda del Hoverboard con un software externo, ya que cuando trata de controlar ambas ruedas haciéndolas mover en un mismo sentido, la velocidad de estas aumenta descontroladamente. Es por ello que, para comprobar si con el Hoverboard del proyecto se tenía la misma problemática, se realizaron dos experimentos. El primero de ellos consistió en tratar de controlar cada rueda independientemente, mientras que en el segundo experimento se quiso controlar ambas ruedas al mismo tiempo. En los siguientes subapartados se explican las conclusiones extraídas de estos experimentos.

3.4.2.1. Control de una rueda

Este experimento consistió en realizar las pruebas controlando una sola rueda a la vez. Se observó que la velocidad del Hoverboard es proporcional a la inclinación de la plataforma. Además, a través de medidas experimentales se encontró que los valores que envían las placas de giroscopio a la controladora principal del Hoverboard están comprendidas en el rango $[-1500, 1500]$, y es en estos límites donde el Hoverboard alcanza su velocidad máxima. También es importante comentar los siguientes puntos:

- Para una misma inclinación, el módulo de la inclinación de ambas placas es el mismo, pero el signo es contrario.
- La velocidad de las ruedas se comporta linealmente al modificar la inclinación de las placas.
- Si la velocidad es negativa, los bytes de la inclinación de la trama se codifican en complemento a 2.

En la figura 3.22 se muestra el comportamiento de la palabra proporcional a la inclinación con respecto a la inclinación de las placas:

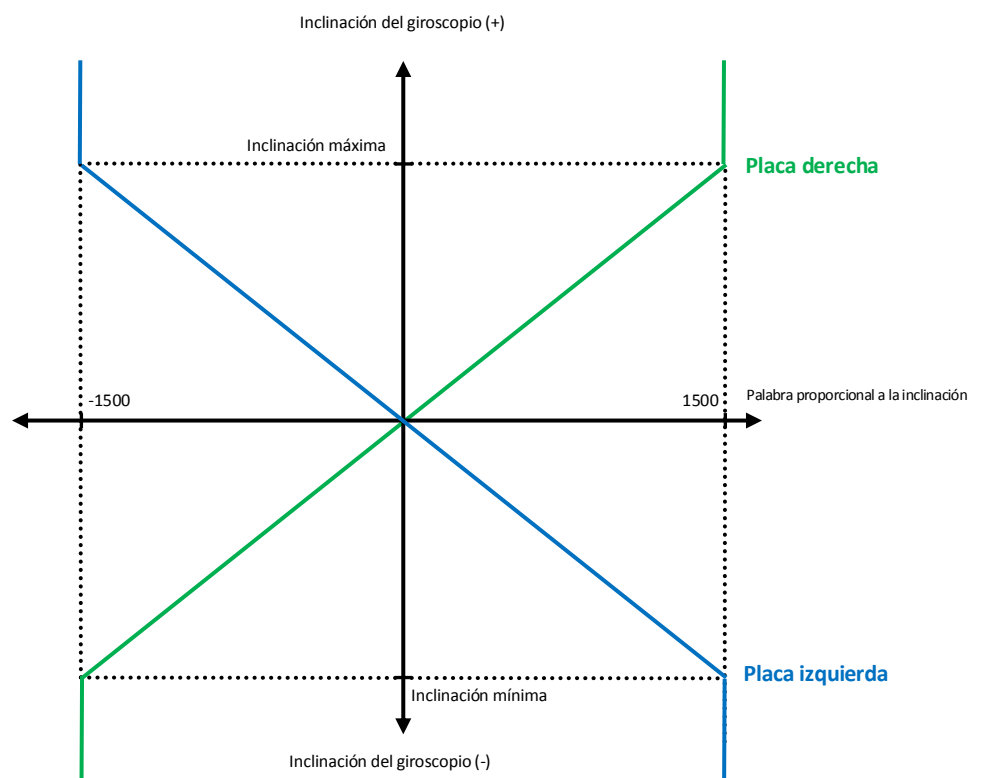


Figura 3.22.-Comportamiento de la palabra proporcional a la inclinación en función de la inclinación de las placas con giroscopio.

En la figura 3.23 se muestra el comportamiento de la velocidad de la rueda del Hoverboard, para distintas consignas de inclinación:

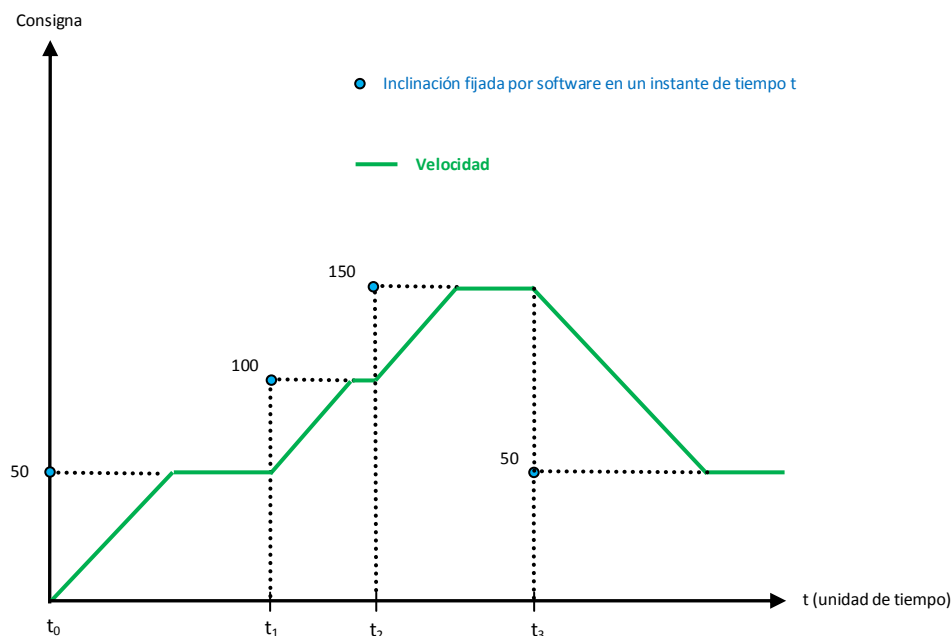


Figura 3.23-Comportamiento de la velocidad de las ruedas del Hoverboard, para distintas consignas de inclinación.

Como se observa en la figura 3.23, cuando se fija una inclinación, la velocidad de las ruedas aumenta hasta llegar a una velocidad que para la controladora, es proporcional a esa inclinación.

Como ejemplo, se supone un escenario donde el usuario del Hoverboard está desplazándose en línea recta, y a velocidad constante. En concreto, se supondrá que el valor de la inclinación enviado por las placas con giroscopio es de 200 para la rueda derecha y de -200 para la rueda izquierda.

La figura 3.24 muestra la trama para controlar la rueda derecha:

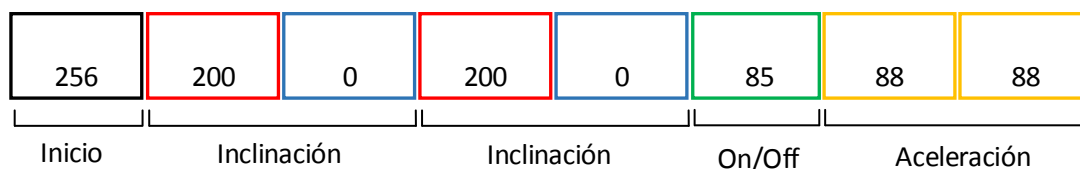


Figura 3.24- Trama de datos que envía Arduino por comunicación UART para mover la rueda derecha a una velocidad proporcional a una inclinación de 200.

La figura 3.25 muestra la trama para controlar la rueda izquierda:

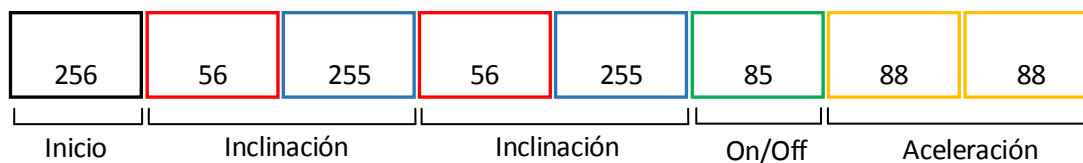


Figura 3.25- Trama de datos que envía Arduino por comunicación serial para mover la rueda izquierda a una velocidad proporcional a una inclinación de -200.

En las tramas de las figuras 3.24 y 3.25 se observa que los valores de inclinación son 0x0068 para la rueda derecha y 0xFF38 para la rueda izquierda (-200 en complemento a 2).

Una vez realizadas las pruebas controlando una sola rueda a la vez, y verificando que se podían controlar sin problemas, se realizaron los experimentos moviendo las dos ruedas al mismo tiempo, con el objetivo de comprobar si ocurría la misma problemática que en la referencia.

3.4.2.2. Control de dos ruedas

Para este segundo experimento, se añadió una segunda placa Arduino Nano, por lo que se prescinde totalmente de las placas con giroscopio, como se representa en la figura 3.26:

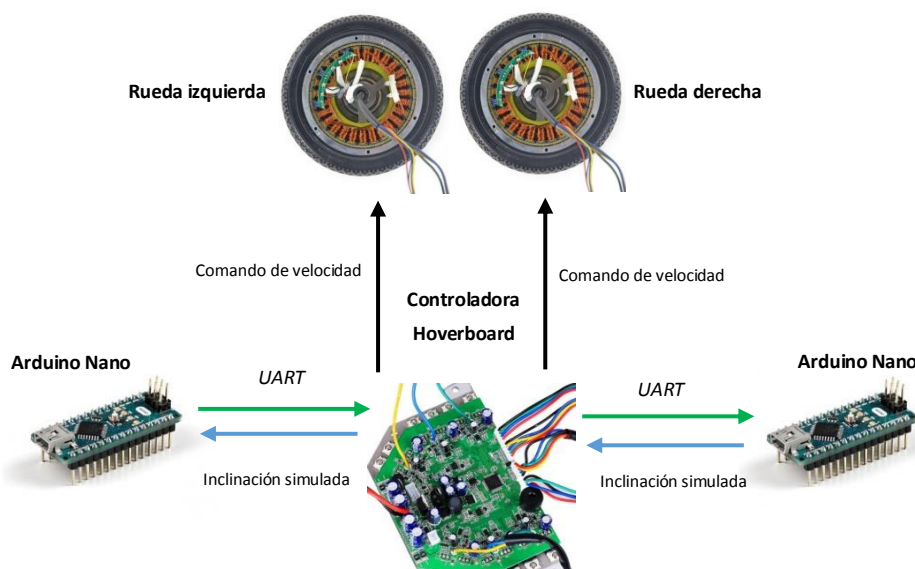


Figura 3.26.- Implementación del experimento con dos ruedas, donde cada placa con giroscopio se ha intercambiado por una placa Arduino Nano.

Este experimento consistió en verificar que el Hoverboard era capaz de realizar movimientos rectilíneos y movimientos de giro sobre su eje central, ya que si se quiere aprovechar el Hoverboard para dar movimiento a una silla de ruedas eléctrica, se tendría que ser capaz de realizar estos movimientos sin problemas.

En la figura 3.27 se muestran los movimientos que se quieren reproducir en el experimento:

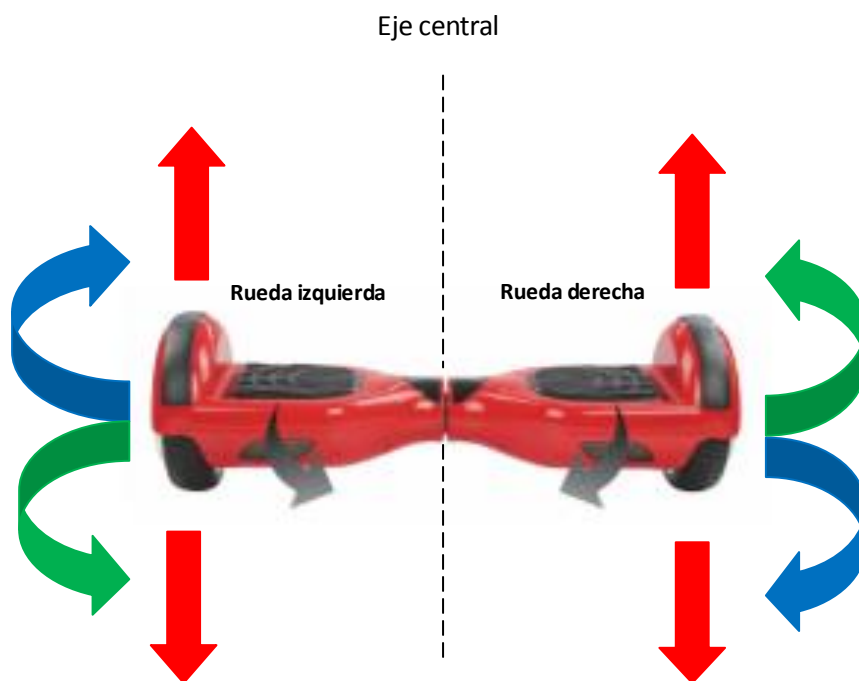


Figura 3.27.- Movimientos que se quieren realizar con el Hoverboard en la segunda prueba.

- **Giro hacia la derecha sobre el eje central:**

En este movimiento, la rueda izquierda se moverá hacia adelante, mientras que la rueda derecha se moverá hacia atrás, a una misma velocidad.

- **Giro hacia la izquierda sobre el eje central:**

Para este movimiento, la rueda derecha se moverá hacia adelante, mientras que la rueda izquierda se moverá hacia atrás, a una misma velocidad.

- **Movimientos rectilíneos:**

En este caso, tanto la rueda izquierda como la derecha se moverán en un mismo sentido y a una misma velocidad.

Los valores necesarios de inclinación enviados desde Arduino Nano para realizar los movimientos representados en la figura 3.27, son los mostrados en la tabla 3.3:

Tabla 3.3.- Valores de inclinación que envían ambas placas Arduino Nano en función de los movimientos que se desean experimentar, representados en la figura 3.27.

	Rueda derecha	Rueda izquierda
Giro a la derecha	0x068	0x068
Giro a la izquierda	0xFF38	0xFF38
Avance	0xFF38	0x068
Retroceso	0x068	0xFF38

Los experimentos de giro, tanto a la izquierda como a la derecha, obtuvieron los resultados esperados. Se podía realizar el giro del Hoverboard respecto al eje central del Hoverboard a cualquier velocidad deseada, simplemente cambiando el valor de los bytes de la inclinación de la trama enviada a la controladora.

No obstante, al probar el desplazamiento lineal (tanto avanzando como retrocediendo), la velocidad de las ruedas aumentaba indefinidamente, fuera cual fuera la consigna de inclinación. Esta problemática, mencionada ya en la referencia, se muestra en la figura 3.28:

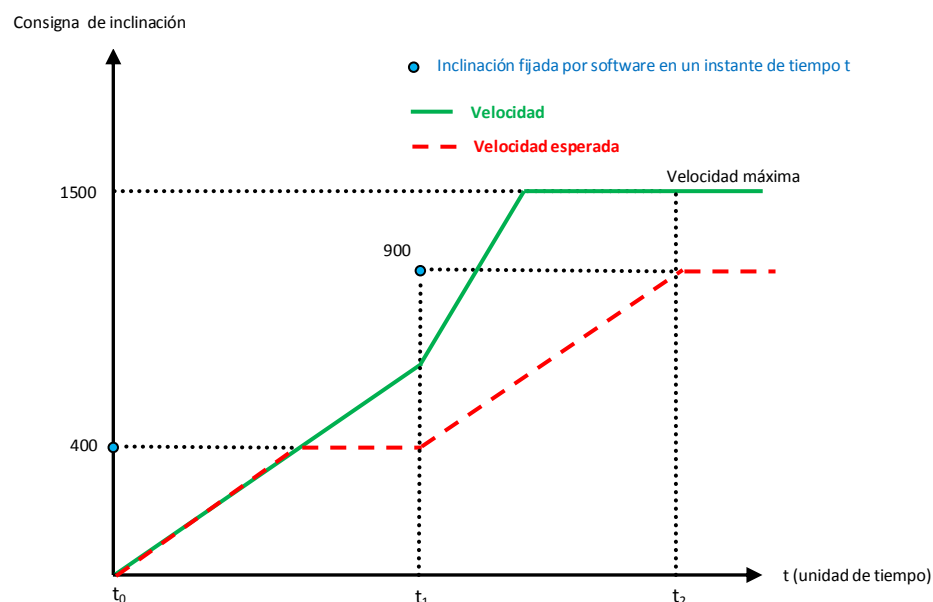


Figura 3.28- Gráfico que muestra el comportamiento de la velocidad en las pruebas realizadas con dos ruedas, moviéndose estas en un mismo sentido. En rojo, el comportamiento de velocidad esperado. En verde, el comportamiento de velocidad actual.

También se observó cómo, a mayor valor de consigna de inclinación, más rápidamente se llega a la velocidad máxima del Hoverboard, tal y como se muestra en la figura 3.29:

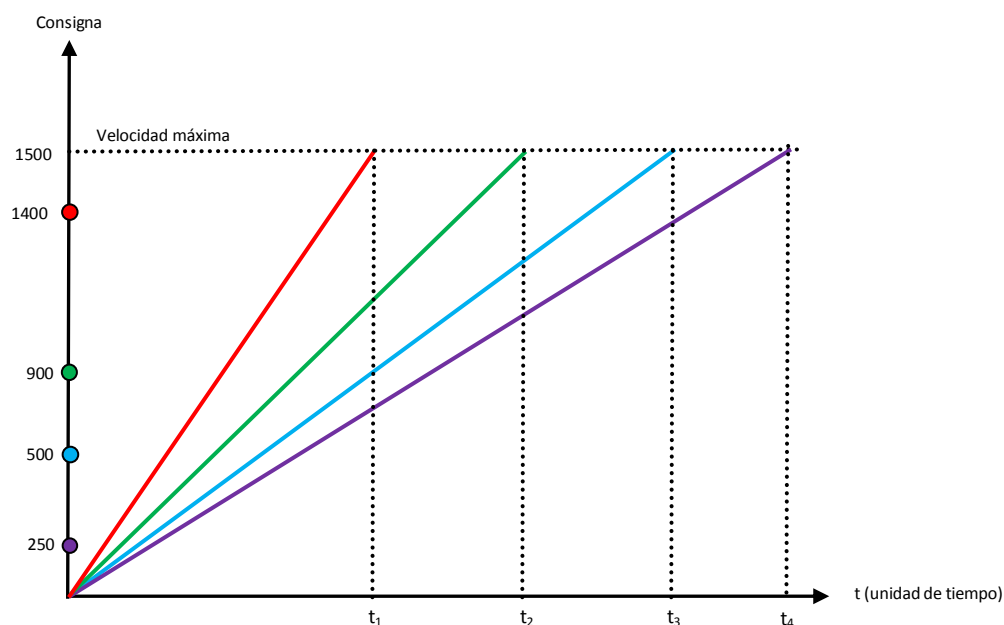


Figura 3.29- Gráfico del comportamiento de la velocidad de las ruedas del Hoverboard. Cuando el valor de la inclinación es mayor (250, 500, 900 y 1400), más rápidamente llega a la velocidad máxima ($t_1 < t_2 < t_3 < t_4$).

3.4.2.3. Hipótesis de la problemática

Se planteó una hipótesis respecto a las causas que provocaban que el Hoverboard acelerase hasta alcanzar su velocidad máxima con una consigna fija de inclinación. Para ello, se explicará brevemente en qué consiste un sistema de péndulo invertido [20], y la relación que tiene este sistema con el comportamiento del Hoverboard.

Un péndulo invertido es un ejemplo popular que se suele encontrar en la literatura de control de sistemas. La característica principal del sistema es que el centro de masa del péndulo se encuentra por encima de su punto de pivote, como muestra la figura 3.30:

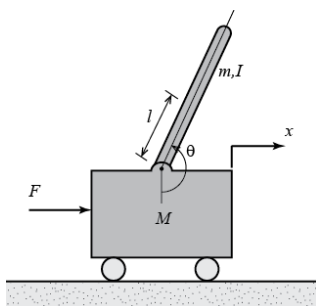


Figura 3.30.- Ejemplo de un sistema de péndulo invertido. [20]

Donde:

- M = Masa del vehículo
- m = masa del péndulo
- l = Distancia al centro de masa del péndulo
- I = Momento de inercia del péndulo
- F = Fuerza aplicada al vehículo
- Θ = Ángulo del péndulo respecto a la vertical

El sistema es inestable en lazo abierto y, en consecuencia, el péndulo caerá si el carro o el vehículo donde está sujeto no se mueve para balancearlo. Además, la dinámica del sistema no es lineal.

El objetivo del control para un péndulo invertido será balancearlo aplicando una fuerza al vehículo donde el péndulo está sujeto.

Si se compara el sistema anterior al del Hoverboard cuando está siendo utilizado por una persona, existe una semejanza (figura 3.31) que indicaría que el sistema persona-Hoverboard funciona de forma similar a como lo haría un sistema de péndulo invertido.

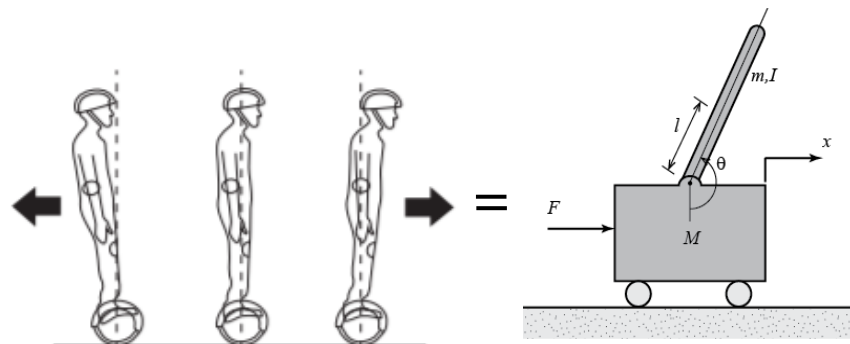


Figura 3.31.- Figura que muestra la similitud física de ambos sistemas. [20]

Dinámicamente, la persona correspondería al péndulo invertido, dado que el centro de masa de la persona se encuentra por encima del punto de pivote (los pies en la plataforma del Hoverboard).

La inclinación de la persona es corregida por el control del Hoverboard, el cual actúa para que las ruedas avancen en la dirección correcta, con el objetivo de compensar la inclinación y evitar que la persona caiga.

La problemática podría estar en las consignas de inclinación que envía el software, las cuales son constantes. Si la consigna es constante, la controladora entiende que la persona se encuentra inclinada constantemente, y, en consecuencia, el Hoverboard acelerará para compensar esa inclinación y evitar que la persona caiga. Aun así, como la controladora no detecta cambios que indiquen que la inclinación disminuye (dado que el valor de inclinación sigue constante), el Hoverboard seguirá acelerando hasta alcanzar la velocidad máxima.

Para estudiar y analizar más exhaustivamente el comportamiento del Hoverboard, se realizó un segundo proceso de ingeniería inversa, el cual se explica en el siguiente subapartado.

3.4.3. Segundo proceso de ingeniería inversa

En este segundo proceso de ingeniería inversa se decidió capturar las tramas enviadas entre las placas con giroscopio y la controladora principal cuando la plataforma está funcionando con una persona encima. Además, ya no se trabajará en vacío, por lo que será la propia persona la que modificará la inclinación de las placas con giroscopio con los pies.

Este apartado se dividirá en tres subapartados: la implementación del experimento en cuestión, la creación del software requerido para capturar las tramas del Hoverboard, y finalmente el análisis de los resultados y las conclusiones del proceso.

3.4.3.1. Implementación del experimento

El experimento consistió en realizar un desplazamiento en línea recta con el Hoverboard a velocidades bajas, medias, y altas. De esta manera, se pretende capturar las tramas de datos en distintas condiciones, ya que la información que envía la placa con giroscopios cuando el Hoverboard se mueve lentamente podría ser diferente a cuando se mueve a más velocidad.

Se capturará tanto la información que envían las placas con giroscopio como la información que reciben de la placa principal, ya que esta última no se conoce hasta ahora y podría contener información importante que ayudaría a entender mejor la comunicación entre placas.

Para capturar los datos enviados entre placas, se conecta cada hilo de transmisión (Tx) y recepción (Rx) de las placas con giroscopio a un Arduino Nano. En total, 4 Arduinos serán

los encargados de capturar la información, y cada Arduino se concentrará en un hub USB de 4 puertos para conectarlo finalmente al ordenador a través de un USB.

El principal motivo por el cual se utilizan 4 Arduinos es porque la comunicación serial de estos es bloqueante por lo que, si con un mismo Arduino se están leyendo las tramas del canal Rx, no se podían leer al mismo tiempo las tramas del canal Tx y, por ello, se perdían muchos datos.

En la figura 3.32 se muestra la implementación realizada para este experimento:

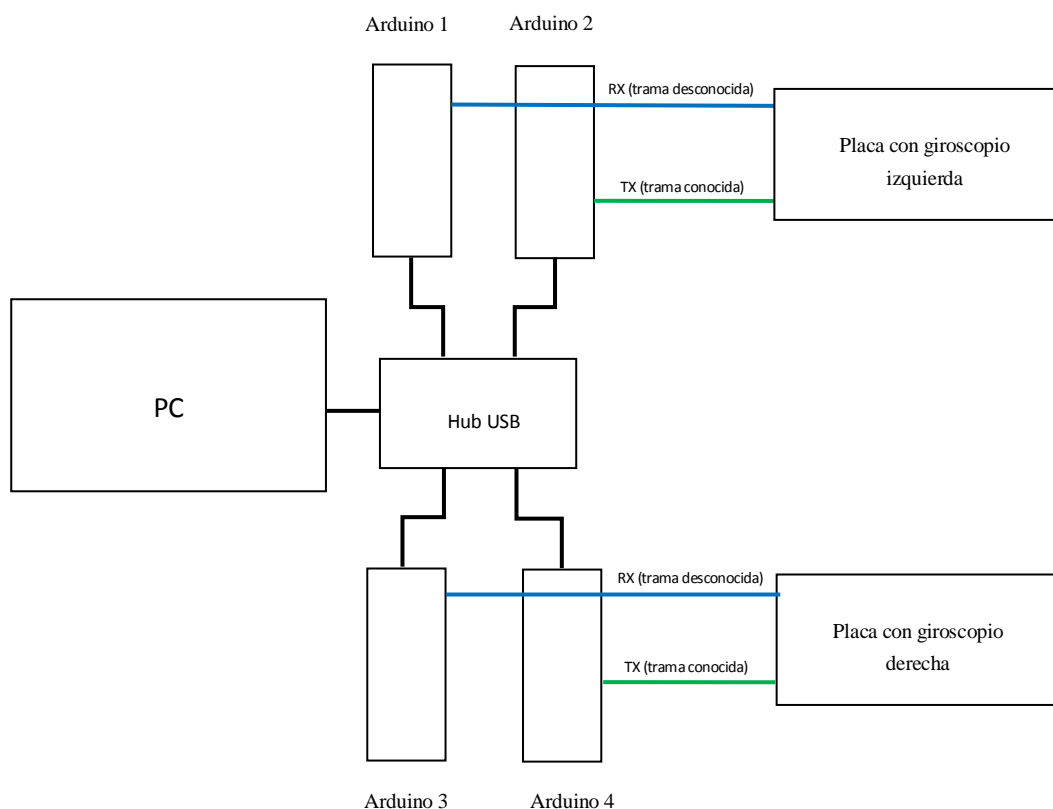


Figura 3.32.- Diagrama de alto nivel de la implementación del experimento. Dos Arduinos se encargaran de leer los canales Tx y los otros dos de los canales Rx.

3.4.3.2. Software

El software del experimento, hecho por un compañero del IRI, se basa en una librería escrita en C++ [A2.1] [A2.2] que permite recibir las tramas de 9 bits de los canales Rx y Tx del Hoverboard. También permite guardar esta información en archivos de texto para facilitar el análisis.

Además, se crearon dos códigos de Arduino, uno para leer los canales de recepción (Rx) [A2.3] y otro para los de transmisión (Tx) [A2.4] de cada una de las placas con giroscopio.

3.4.3.3. Análisis de los datos capturados

Toda la información recibida por las placas Arduino Nano durante el experimento fue guardada en archivos de texto. La figura 3.33 muestra un fragmento de las tramas enviadas por las placas con giroscopio a la controladora principal del Hoverboard:

1256 -	201	255	201	255	85	79	79
1257 -	201	255	201	255	85	79	79
1258 -	200	255	200	255	85	79	79
1259 -	200	255	200	255	85	79	79
1260 -	199	255	199	255	85	79	79
1261 -	197	255	197	255	85	79	79
1262 -	196	255	196	255	85	80	80
1263 -	195	255	195	255	85	79	79
1264 -	192	255	192	255	85	80	80
1265 -	190	255	190	255	85	80	80
1266 -	189	255	189	255	85	80	80
1267 -	189	255	189	255	85	80	80
1268 -	188	255	188	255	85	80	80
1269 -	186	255	186	255	85	80	80
1270 -	186	255	186	255	85	81	81
1271 -	185	255	185	255	85	81	81
1272 -	184	255	184	255	85	81	81
1273 -	182	255	182	255	85	80	80
1274 -	181	255	181	255	85	80	80
1275 -	179	255	179	255	85	81	81
1276 -	178	255	178	255	85	81	81
1277 -	176	255	176	255	85	81	81
1278 -	175	255	175	255	85	80	80
1279 -	174	255	174	255	85	81	81
1280 -	173	255	173	255	85	81	81
1281 -	172	255	172	255	85	81	81
1282 -	171	255	171	255	85	80	80
1283 -	169	255	169	255	85	80	80
1284 -	168	255	168	255	85	80	80
1285 -	168	255	168	255	85	80	80
1286 -	166	255	166	255	85	80	80
1287 -	166	255	166	255	85	80	80
1288 -	165	255	165	255	85	79	79
1289 -	165	255	165	255	85	79	79

Figura 3.33.- Fragmento de datos capturados de la placa con giroscopio, específicamente la placa del lado de la batería.

Seguidamente, las tramas capturadas como la de la figura 3.33 se graficaron con MATLAB, utilizando el código del anexo [A2.5], para realizar un análisis más exhaustivo de la información. Se obtuvieron las siguientes gráficas, a baja, media y alta velocidad, y para cada placa de giroscopio:

Placa cercana a la batería

- Velocidad baja

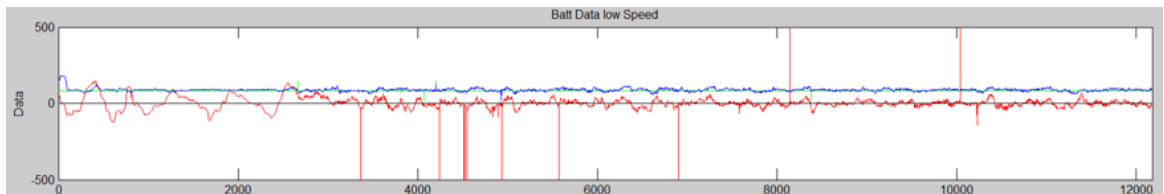


Figura 3.34.- Análisis gráfico de los datos enviados por la placa cercana a la batería en el ensayo a velocidad baja.

- Velocidad media

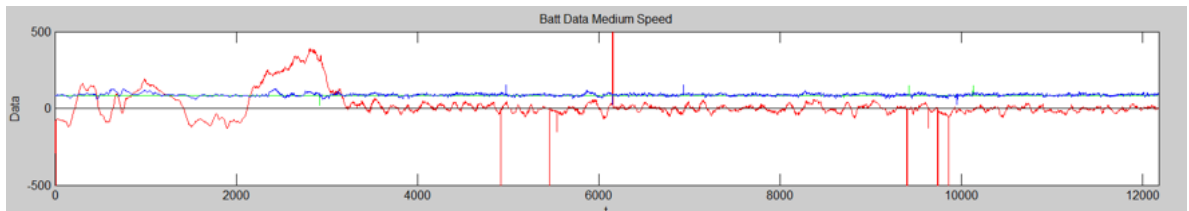


Figura 3.35.- Análisis gráfico de los datos enviados por la placa cercana a la batería en el ensayo a velocidad media.

- Velocidad alta

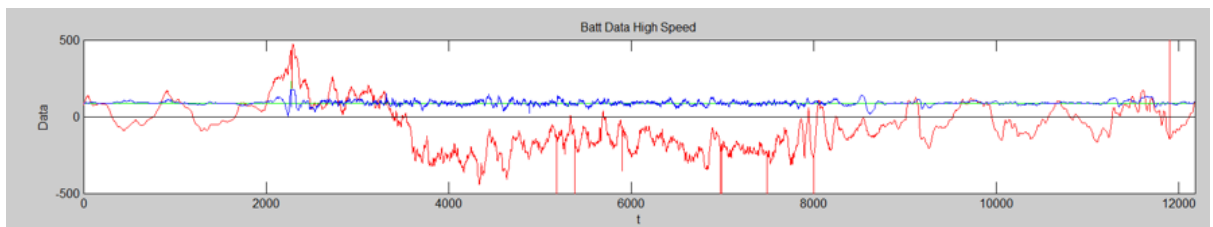


Figura 3.36.- Análisis gráfico de los datos enviados por la placa cercana a la batería en el ensayo a velocidad alta.

Placa cercana a la controladora

- Velocidad baja

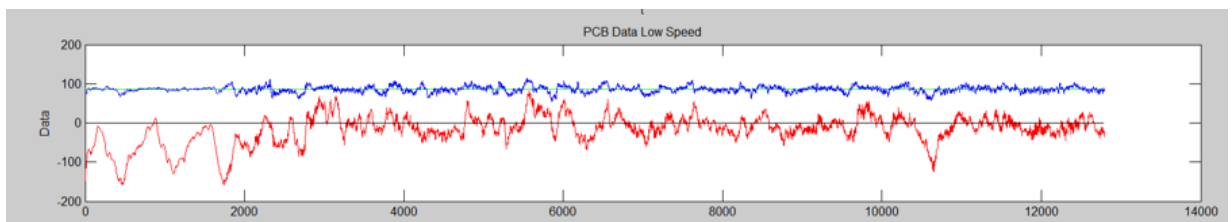


Figura 3.37.- Análisis gráfico de los datos enviados por la placa cercana a la controladora en el ensayo a velocidad baja.

- Velocidad media

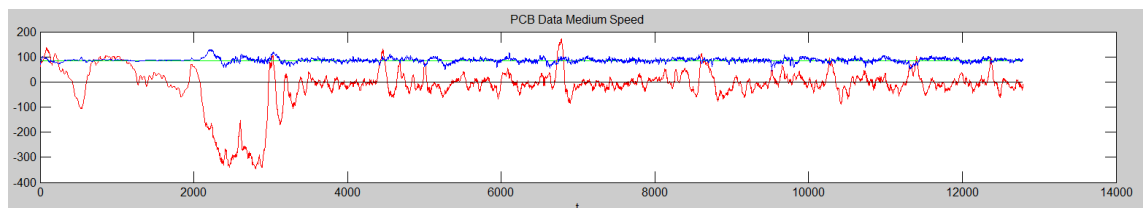


Figura 3.38.- Análisis gráfico de los datos enviados por la placa cercana a la controladora en el ensayo a velocidad media.

- Velocidad alta

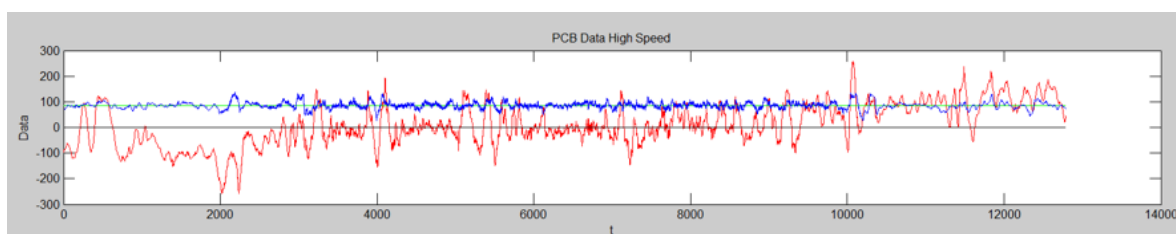


Figura 3.39.- Análisis gráfico de los datos enviados por la placa cercana a la controladora en el ensayo a velocidad alta.

En las gráficas anteriores, el eje de abscisas corresponde al tiempo del ensayo en milisegundos, y el eje de ordenadas, corresponde al valor de los datos capturados, en base decimal.

A partir del análisis de las gráficas anteriores se pudieron extraer las siguientes conclusiones:

- La trama verde corresponde al valor que envían las placas con giroscopio a la controladora informando de si hay alguien montado en el Hoverboard. Como es de esperar, siempre es 85, ya que durante el experimento la persona está montada en el patinete.
- La trama azul corresponde al valor proporcional a la aceleración angular con respecto al eje central del Hoverboard, y se observa como oscila alrededor de 85. La principal hipótesis sobre por qué este valor oscila es debido a que la persona que está utilizando el Hoverboard no se mueve en línea recta perfecta, sino que va corrigiendo el movimiento con los pies y, en consecuencia, los giroscopios detectan cierta aceleración angular.

- La trama en rojo corresponde a las consignas que envían las placas con giroscopio a la controladora principal del Hoverboard.

Si se observa la figura 3.40, correspondiente al ensayo a velocidad alta, la primera zona, marcada con un círculo verde, correspondería a una aceleración del sistema. Luego, la parte central, marcada con un círculo negro, correspondería a cuando el sistema ha llegado a la velocidad establecida por las placas con giroscopio. Además, en este punto la consigna oscila alrededor de 0. Finalmente, una tercera zona, marcada con un círculo naranja, donde el sistema desacelera.

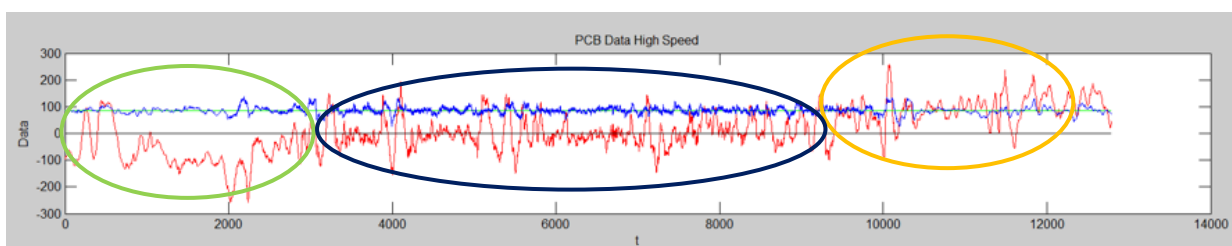


Figura 3.40.- Comportamientos del sistema en el ensayo a velocidad alta. En verde, la aceleración del sistema. En negro, cuando el sistema ha llegado a la velocidad deseada. En naranja, la desaceleración del sistema.

Las observaciones realizadas con respecto al comportamiento de la consigna (trama roja) se fijaron como principal hipótesis para resolver el problema de embalamiento de velocidad que ocurría al controlar ambas ruedas del Hoverboard en un mismo sentido.

3.4.3.4. Control de aceleración

Se modificó el código realizado anteriormente [A1.1], de tal manera que ahora deben haber pulsos de consigna para poder tener cambios en la velocidad de las ruedas. Además, una vez se llega a la velocidad deseada, la consigna oscila alrededor de 0, como se muestra en la figura 3.41:

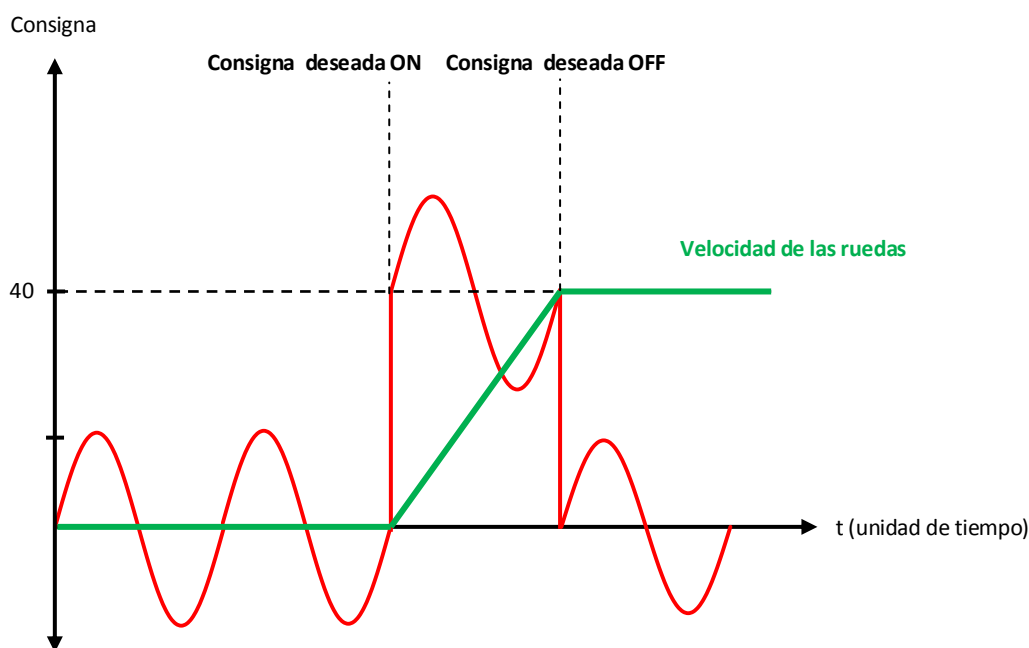


Figura 3.41.- Comportamiento de la consigna mandada a la controladora (en rojo) con la nueva modificación del código. La velocidad de las ruedas responde ahora a pulsos de consigna.

A pesar de haber solucionado el problema de embalamiento con esta modificación, finalmente no se implementaron las oscilaciones en la consigna que muestra la figura 3.41.

Se comprobó experimentalmente como, fijando la consigna a 0 sin oscilar (mientras no haya consigna), y creando un pulso en el momento que llega la consigna del joystick, el control de aceleración también funcionaba correctamente.

De esta manera, cuando el Hoverboard acelerase para llegar a la consigna fijada por software y la sobrepase, debido al problema de embalamiento de velocidad, se fijará rápidamente una consigna nula para indicar a las ruedas que no aceleren más.

Esta hipótesis se validó experimentalmente con el programa del anexo [A2.6]. Con este programa se modifica la consigna enviando los pulsos de aceleración y desaceleración con un

joystick conectado a Arduino. En la figura 3.42 se muestra el comportamiento de la velocidad de las ruedas con el nuevo software diseñado, siguiendo la consigna fijada por el joystick:

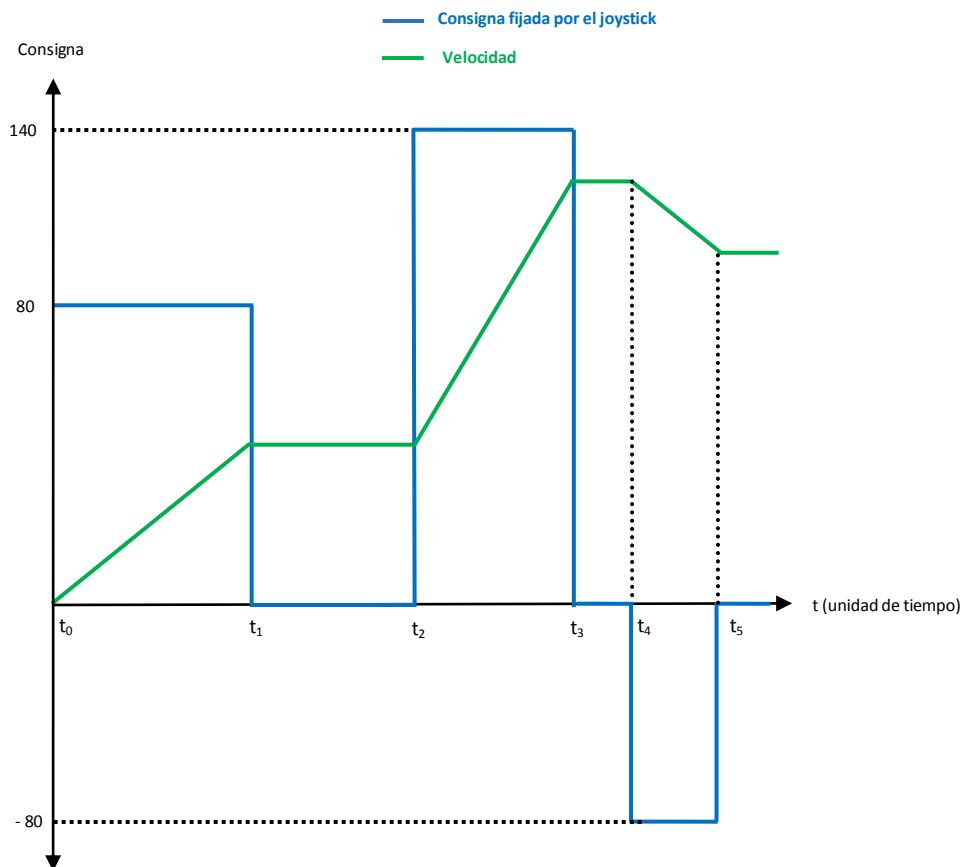


Figura 3.42.- Gráfico del comportamiento de la velocidad de las ruedas, siguiendo las consignas del joystick conectado a Arduino. Ahora, solo hay cambios en la velocidad cuando hay pulsos de consigna.

Intervalo t_0 - t_1 : En este intervalo el joystick fija una consigna igual a 80. La controladora principal ordena a las ruedas que aumenten la velocidad.

Intervalo t_1 - t_2 : En este intervalo el joystick fija una consigna nula (posición central). La controladora principal indica a las ruedas que mantengan la velocidad.

Intervalo t_2 - t_3 : En este intervalo el joystick fija una consigna de 140. La controladora principal ordena a las ruedas que aumente la velocidad. La aceleración es mayor que en intervalo t_0 - t_1 , ya que el valor de inclinación es también mayor.

Intervalo t_3 - t_4 : En este intervalo el joystick fija una consigna nula (posición central). La controladora principal indica a las ruedas que mantengan la velocidad.

Intervalo t_4 - t_5 : En este intervalo el joystick fija una consigna igual a -80. La controladora principal indica a las ruedas que disminuyan la velocidad.

Concluyendo, con este control de aceleración se consiguió controlar ambas ruedas del Hoverboard a la vez, solucionando el problema de embalamiento de velocidad que indicaba la referencia. No obstante, este tipo de control no es válido para la aplicación final que se requiere en el proyecto, ya que se necesita implementar un control de velocidad.

Por lo tanto, para poder diseñar un control de velocidad, será necesario obtener una realimentación de velocidad de las ruedas del Hoverboard.

3.5. Realimentación de velocidad

Para implementar el control de velocidad deseado, habría que obtener una realimentación de la velocidad de las ruedas para poder establecer una acción de control en la consigna enviada a la controladora del Hoverboard. El control que se desea hacer se muestra la figura 3.43:

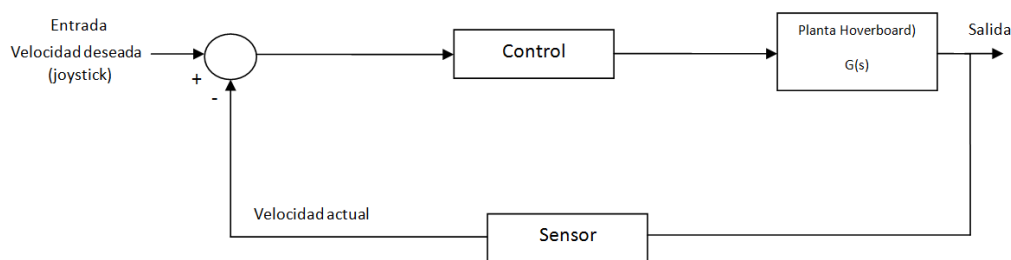


Figura 3.43.- Control de velocidad requerido para el proyecto.

Para obtener la lectura de velocidad deseada e implementarla en el control, se estudiaron dos opciones distintas, explicadas en las siguientes secciones.

3.5.1. Ratón óptico como encoder

La primera solución que se optó para obtener una lectura de velocidad de las ruedas fue aprovechar un ratón óptico de ordenador, como los que se pueden encontrar en el mercado.

Uno de los principales motivos por el cual se quiso reutilizar un ratón óptico para leer la velocidad de las ruedas fue su bajo precio, y dado que el proyecto requiere reducir los costes al máximo era una opción muy interesante.

Para ello, se creó la librería escrita en C++ [A2.7] [A2.8] y la API pública [A2.9]. Con esta API es posible leer la posición relativa en los ejes X e Y del plano donde se mueve el ratón, además de obtener la velocidad en esos mismos ejes. También es posible leer esa información de más de un ratón a la vez. [21] [22]

Se realizaron distintos ensayos con el objetivo de comprobar si la lectura del ratón óptico era fiable. Para ello, se procedió experimentalmente a desplazar el ratón a diferentes distancias de la superficie, tanto en el eje X como en el eje Y. Los resultados obtenidos del ensayo se muestran en las tablas 3.4 y 3.5:

Para 20 cm de desplazamiento en el eje X de la superficie:

Tabla 3.4. - Lecturas obtenidas para un desplazamiento de 20 cm con el ratón en el eje X de la superficie.

Número de ensayo	Distancia medida con el ratón (cm)
1	19,74
2	19,80
3	19,87
4	19,85
5	20,05

Para 5 cm de desplazamiento en el eje Y de la superficie:

Tabla 3.5. - Lecturas obtenidas para un desplazamiento de 5 cm con el ratón en el eje Y de la superficie.

Número de ensayo	Distancia medida con el ratón (cm)
1	5,15
2	5,16
3	4,99
4	5,09
5	5,26

A partir de los resultados mostrados en las tablas 3.4 y 3.5, se observa como la lectura de desplazamiento del ratón es de confianza.

No obstante, su implementación en el Hoverboard era compleja, ya que la lectura del ratón depende mucho de la superficie donde se mueve y, para ello, había que diseñar una estructura que fijase muy bien la electrónica del ratón en las ruedas para obtener una lectura de la velocidad de confianza. La figura 3.44 muestra un prototipo de este soporte para tratar de fijar la electrónica del ratón óptico a las ruedas del Hoverboard:



Figura 3.44.- Soporte creado con impresora 3D para fijar la electrónica del ratón óptico.

A pesar de haber creado el driver para leer la posición y la velocidad del ratón óptico, se decidió recurrir a otra solución para leer la velocidad de las ruedas. Los motores del Hoverboard contienen en su interior sensores Hall. Si de algún modo se podían aprovechar estos sensores para la lectura de la velocidad, se disminuirían tanto los costes como la complejidad del proyecto, dado que no habría que adquirir ningún sensor extra.

Finalmente, se optó por la segunda opción, ya que tanto por precio como por fiabilidad en la lectura era más viable que utilizar el ratón óptico como encoder. En el siguiente apartado se explica todo el procedimiento llevado a cabo para obtener una lectura de velocidad de las ruedas aprovechando los sensores Hall del Hoverboard.

3.5.2. Sensores Hall

Los motores DC sin escobillas [24] del Hoverboard disponen de 3 sensores Hall embebidos en su interior, como muestra la figura 3.45:

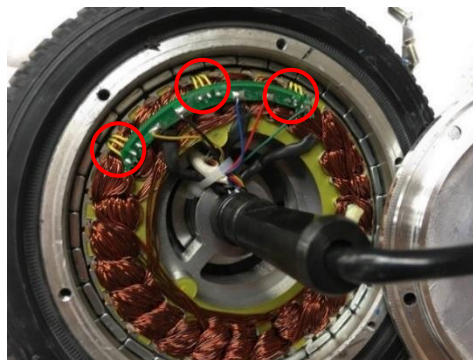


Figura 3.45.- Interior de una de las ruedas del Hoverboard, donde se encuentra un motor DC sin escobillas y tres sensores Hall, marcados en la imagen con un círculo.

La figura 3.46 muestra el diagrama eléctrico aproximado de uno de los motores del Hoverboard, con el conexionado de los interruptores electrónicos (MOSFETS), los tres sensores Hall A, B y C, y sus señales de conmutación:

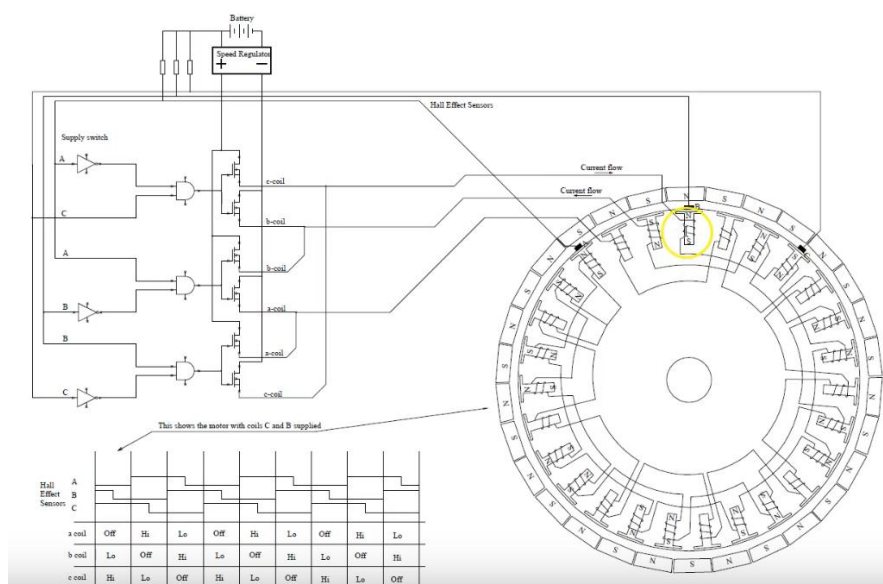


Figura 3.46.- Esquemático simplificado del motor DC sin escobillas localizado en cada una de las ruedas del Hoverboard. [25]

En la figura 3.47 se muestra la secuencia de energizado de las bobinas del motor con respecto al estado de los sensores Hall [25]:

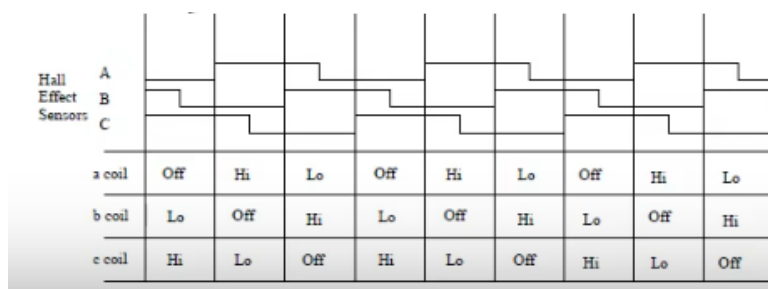


Figura 3.47.- Secuencia de energizado de las bobinas del motor del Hoverboard con respecto al estado de los sensores Hall A, B y C. [25]

3.5.2.1. Proceso de ingeniería inversa

En primer lugar, se capturó con la ayuda de un osciloscopio la señal de uno de los sensores Hall del motor del Hoverboard, como se muestra en la figura 3.48:

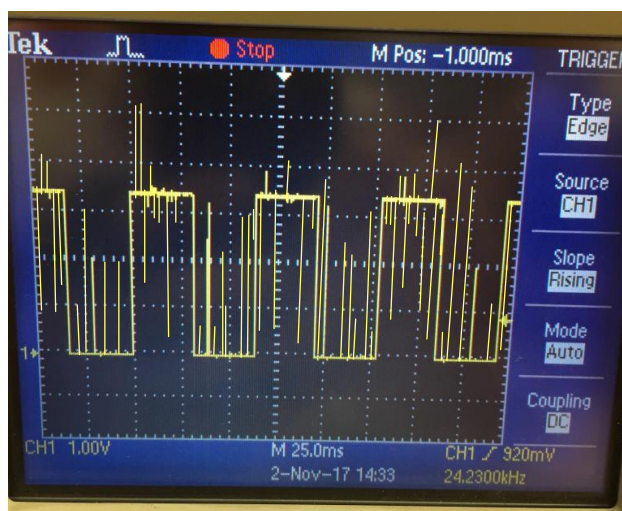


Figura 3.48.- Señal con ruido de uno de los tres sensores Hall del motor capturada con un osciloscopio.

Se aprecia como hay ruido de alta frecuencia en la señal del sensor. Según la referencia [26], en la placa principal del Hoverboard se encuentra un filtro RC paso bajo encargado de filtrar la señal anterior antes de ser procesada por el microcontrolador del Hoverboard. Este filtro se representa en la figura 3.49:

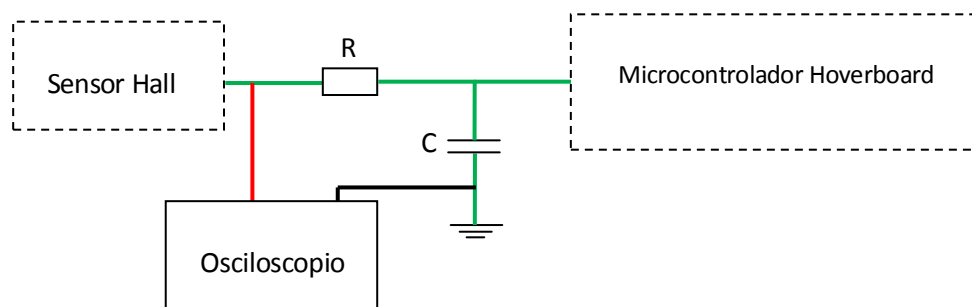


Figura 3.49.- Conexión del osciloscopio en el circuito del filtro paso bajo. El valor de los componentes pasivos es desconocido.

Al haber capturado la señal con el osciloscopio antes del filtro, como se observa en la figura 3.49, la señal no aparece filtrada.

Si se quiere utilizar la señal de los sensores Hall como realimentación en el control de velocidad, hay que eliminar el ruido que aparece en la figura 3.48, ya que se obtendrían lecturas erróneas.

Una posible solución sería adquirir la señal de los sensores después del filtrado, es decir, obteniendo las señales de los sensores Hall directamente de la placa de circuito impreso de la controladora principal, pero esta solución se evitó dado que cabía la posibilidad de dañar la placa principal al realizar soldaduras y manipulaciones en ella.

Una solución sencilla sería diseñar un filtro paso bajo para filtrar la señal de los sensores Hall. Aun así, esta solución no era viable ya que al conectar el filtro paso bajo a la entrada del microcontrolador se añadiría una impedancia al circuito que distorsionaría la señal de los sensores Hall.

Por consiguiente, se ideó el circuito de la figura 3.50, con el objetivo de filtrar la señal de los sensores:

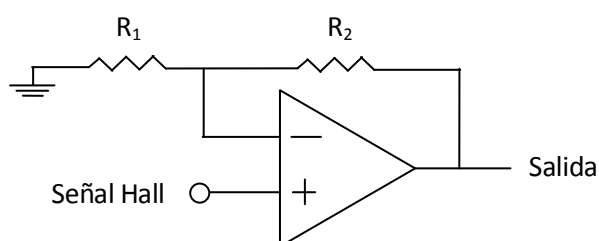


Figura 3.50.- Amplificador no inversor diseñado para filtrar la señal proveniente de los sensores Hall.

El circuito de la figura 3.50 se corresponde a un amplificador operacional en configuración no inversora, con el cual se pretende:

- Aislar impedancias entre la señal del sensor Hall y Arduino. De esta manera, la señal de los sensores Hall no se verá modificada.
- Amplificar la señal de los sensores Hall de 3,3 V a 5 V, ya que los pines digitales de Arduino trabajan con un umbral de 0 V a 2,5 V para un estado lógico bajo, y de 3,5 V a 5 V para un estado lógico alto. La figura 3.51 muestra estos umbrales de tensión para los pines digitales de Arduino:

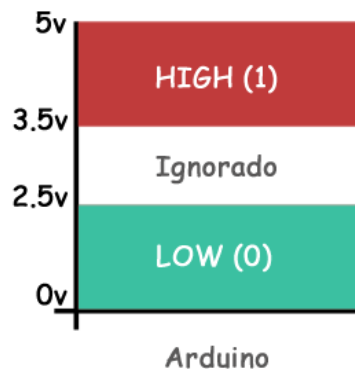


Figura 3.51.- Umbrales de tensión de los pines digitales de Arduino. [23]

- Filtrar la frecuencia de la señal, disminuyendo su ancho de banda.

Se elige una ganancia elevada para saturar la señal de salida del amplificador, pero no lo suficientemente alta como para reducir el ancho de banda lo suficiente para distorsionar la salida.

La ganancia de amplificador operacional en configuración no inversora, es la definida en la expresión 3.1:

$$G = 1 + \frac{R_2}{R_1} \quad (3.1)$$

Se escoge una resistencia R_1 de 1 k Ω y R_2 de 11 k Ω . Luego, la ganancia del amplificador no inversor es la siguiente:

$$G = 1 + \frac{11 \text{ k}\Omega}{1 \text{ k}\Omega} = 12 \quad (3.2)$$

La constante ganancia – ancho de banda, definida como muestra la expresión 3.3:

$$k = G \cdot \omega \quad (3.3)$$

Donde $k = 10 \text{ MHz}$ [27] y $G = 12$. Luego, el valor del ancho de banda es:

$$\omega = \frac{k}{G} = \frac{10 \text{ MHz}}{12} = 833,33 \text{ kHz} \quad (3.4)$$

Después de implementar el circuito ideado, la señal quedó perfectamente filtrada, tal y como se muestra en la figura 3.52:

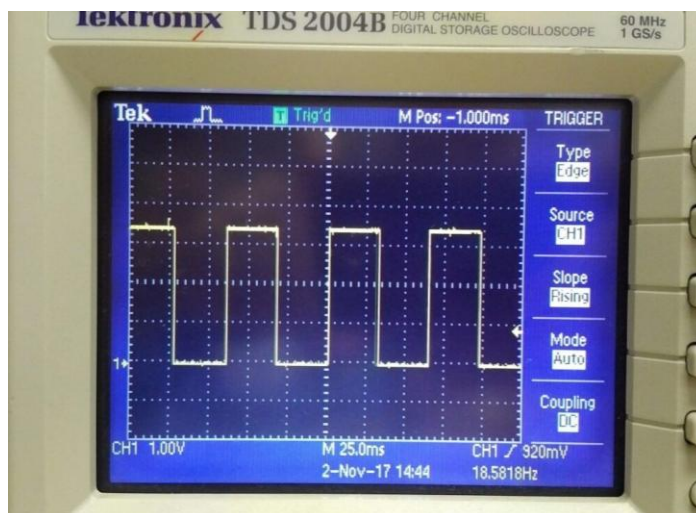


Figura 3.52.- Señal capturada de uno de los sensores Hall, totalmente filtrada por el circuito 3.50.

Una vez mejorada la calidad de la señal de los sensores Hall gracias al circuito diseñado, se realizaron los cálculos correspondientes para dar con una expresión que permitiese obtener la velocidad angular de las ruedas. En el siguiente subapartado se muestran estos cálculos.

3.5.2.2. Lectura de la magnitud de la velocidad

Por definición, la velocidad angular se define como el ángulo recorrido por unidad de tiempo:

$$\omega = \frac{d\theta}{dt} \quad (3.5)$$

Para poder calcular la velocidad angular de las ruedas se medirá el tiempo transcurrido entre los flancos correspondientes de la señal de los sensores de efecto Hall, los cuales se crean en las transiciones entre imanes. Eso es, el ángulo recorrido entre dos flancos de pulso de sensor Hall.

Además, sabiendo que los motores del Hoverboard contienen 30 imanes, tal y como muestra la figura 3.53, el ángulo que abarca cada uno de estos imanes es el calculado en la expresión 3.6:

$$\text{Ángulo}_{\text{imán}} = \frac{360^\circ}{30 \text{ imanes}} = 12^\circ/\text{iman} \quad (3.6)$$



Figura 3.53.- Dibujo del motor DC sin escobillas de una de las ruedas del Hoverboard.

Aproximadamente (dado que los imanes no están colocados perfectamente), cada 12° recorridos hay un flanco de la señal del sensor Hall. Por lo tanto, la expresión para calcular la velocidad angular quedará de la siguiente manera:

$$\omega = \frac{12^\circ}{T \text{ (s)}} \quad (3.7)$$

Donde T es el tiempo transcurrido entre flancos de la señal del sensor Hall, como se muestra en la figura 3.54:

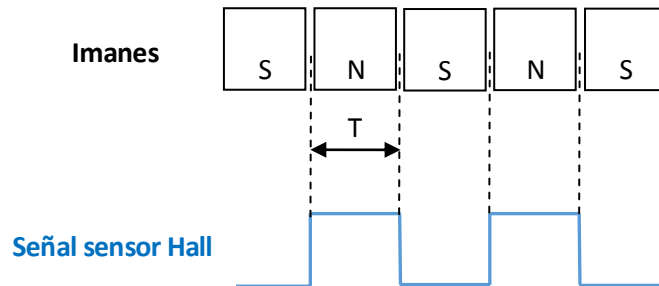


Figura 3.54.- En las transiciones de distintos imanes (N y S) es donde se crean los flancos del sensor Hall.

Se realiza un cambio de unidades para expresarlo en radianes por segundo:

$$\omega = \frac{12^\circ}{T(s)} \cdot \frac{\pi}{180^\circ} = \frac{0,209 \text{ rad}}{T(s)} \quad (3.8)$$

Un aspecto importante a considerar es la frecuencia con la que se realizará la lectura de la señal, ya que dependiendo de esta frecuencia de actualización el control podrá compensar más rápidamente la velocidad de las ruedas.

Para conocer con qué frecuencia se pueden obtener los pulsos, se realizan los siguientes cálculos:

Primeramente, se calcula el perímetro de la rueda, como muestra la expresión 3.9:

$$\text{Perímetro}_{\text{rueda}} = 2 \cdot \pi \cdot r = 2 \cdot \pi \cdot 127 \text{ mm} = 800 \text{ mm} \quad (3.9)$$

Donde r es el radio de la rueda, conocido.

El tiempo T_{vuelta} , en segundos, que tardará la rueda en realizar una vuelta:

$$T_{\text{vuelta}}(s) = \frac{\text{Perímetro}_{\text{rueda}}}{V_{\text{tras}}} = \frac{800 \text{ mm}}{V(\text{mm/s})} = \frac{0,8 \text{ m}}{V(\text{m/s})} \quad (3.10)$$

Finalmente, sabiendo que el motor tiene 15 pares de polos, se puede intuir que en una vuelta completa de la rueda se habrán producido 30 pulsos de sensor Hall.

En consecuencia, la frecuencia f_{pulsos} con la que se producen los pulsos:

$$f_{\text{pulsos}}(\text{Hz}) = \frac{\text{pulsos}}{s} = \frac{\# \text{ pulsos por vuelta}}{T_{\text{vuelta}}} = \frac{30}{0,8 \frac{\text{m}}{V(\text{m/s})}} = \frac{30 \cdot V(\text{m/s})}{0,8 \text{ m}} = \frac{37,5}{\text{m}} \cdot V(\text{m/s}) \quad (3.11)$$

A partir de la expresión (3.11), se puede deducir que la frecuencia de los pulsos aumentará a medida que aumenta la velocidad de las ruedas. La tabla 3.4 muestra algunos de estos valores de frecuencia en función de esta velocidad:

Tabla 3.6.- Tabla donde se muestra los distintos valores de la frecuencia de los pulsos de la señal en función de la velocidad lineal de la rueda.

Velocidad lineal de la rueda	Frecuencia de los pulsos
0,1 m/s	3,75 Hz
0,2 m/s	7,5 Hz
0,6 m/s	22,5 Hz
1,5 m/s	56,25 Hz

Al trabajar con velocidad muy bajas (0,1 m/s o 0,2 m/s), se observa como la frecuencia de los pulsos es muy baja, por lo que se considera que el control de velocidad sería muy poco robusto en esos casos. En consecuencia, se cree conveniente combinar las tres señales de los sensores Hall en una única señal, para así aumentar la frecuencia de los pulsos.

Para ello, se realizó el circuito con puertas lógicas XOR de la figura 3.55:

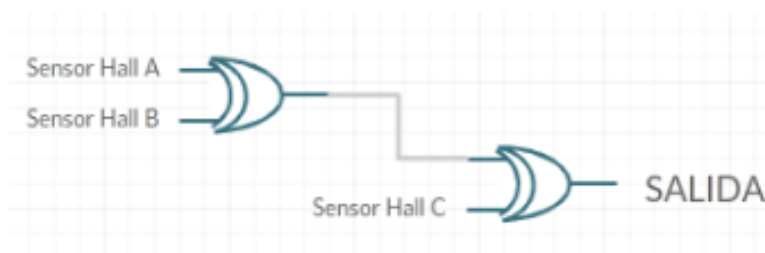


Figura 3.55.- Circuito realizado con puertas lógicas XOR para combinar las tres señales de los sensores Hall en una única señal.

Teniendo en cuenta la función lógica XOR, se puede ver que la señal de salida del circuito de la figura 3.55 tendrá todos los flancos de las tres señales Hall, tal y como se observa en la figura 3.56:

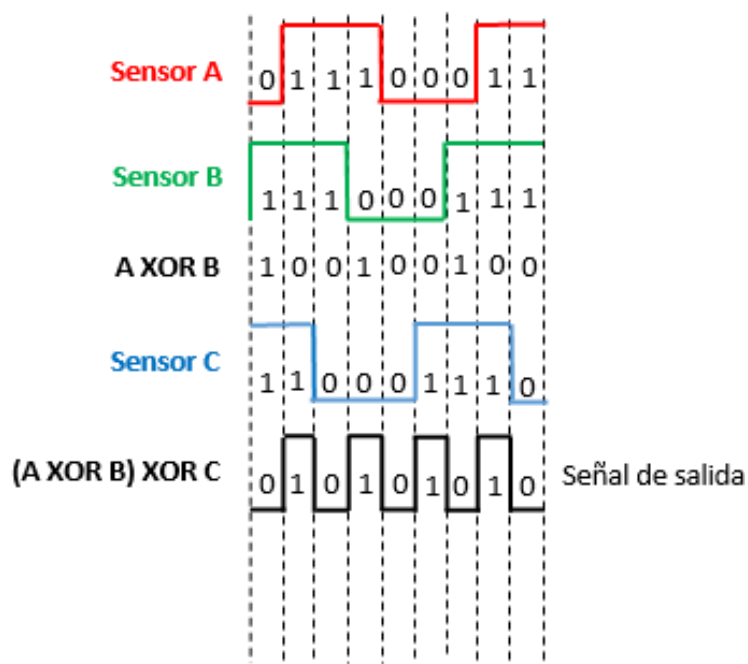


Figura 3.56.- Salida del circuito (en negro) al combinar las señales A, B y C con el circuito 3.55.

Con este circuito se consigue triplicar la frecuencia de actualización de la velocidad de las ruedas:

$$f_{pulsos}(Hz) = \frac{37,5}{m} \cdot V(m/s) \cdot 3 \text{ sensores} = \frac{112,5}{m} \cdot V(m/s) \quad (3.12)$$

Como ejemplo, si las ruedas se mueven a una velocidad lineal de 0,2 m/s, la frecuencia de actualización de velocidad será de 22,5 Hz.

También la expresión (3.8) se modifica al utilizar 3 sensores, como muestra la expresión 3.13:

$$\omega = \frac{\frac{12^\circ}{T(s)} \cdot \frac{\pi}{180^\circ}}{3 \text{ sensores}} = \frac{0,069 \text{ rad}}{T(s)} \quad (3.13)$$

Una vez encontrada una expresión con la cual calcular la velocidad, además de haber mejorado la frecuencia de actualización de los pulsos provenientes de la señal de los sensores Hall, se realizó un programa con Arduino, encontrado en el anexo [A2.10], con el objetivo de leer la velocidad angular de las ruedas por software.

Las primeras pruebas del código se realizaron capturando los flancos ascendentes y descendientes de la señal, como muestra la figura 3.57, conectando la salida del circuito lógico al pin 3 de Arduino Nano (los pines 2 y 3 de Arduino Nano se utilizan para recibir interrupciones externas):

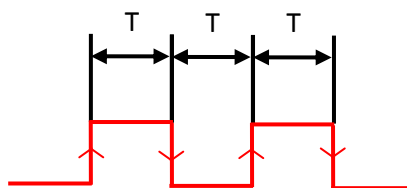


Figura 3.57.- Lectura del tiempo transcurrido entre flancos de la señal de salida del circuito lógico diseñado

Este valor de tiempo T se sustituirá en la expresión (3.13) para obtener la velocidad angular de la rueda.

Una vez implementado el código en Arduino, se capturaron los pulsos de la señal con el osciloscopio bajo distintas velocidades a fin de observar su comportamiento. La velocidad en cada prueba se obtuvo experimentalmente con un cronómetro, calculando el tiempo que la rueda tardaba en realizar una vuelta. A continuación se muestran algunos de los resultados experimentales obtenidos:

- Prueba 1: Velocidad estimada $w = 1 \text{ rad/s}$

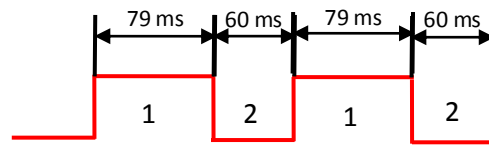


Figura 3.58.- Tiempo transcurrido entre flancos de la señal del sensor Hall en la primera prueba, a 1 rad/s

Velocidad en 1: $w = 0,87 \text{ rad/s}$

Velocidad en 2: $w = 1,15 \text{ rad/s}$

- Prueba 2: Velocidad estimada $w = 1,8 \text{ rad/s}$

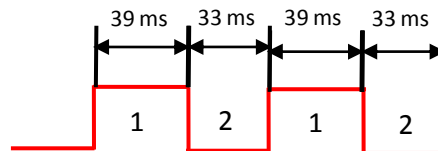


Figura 3.59.- Tiempo transcurrido entre flancos de la señal del sensor Hall en la segunda prueba, a $1,8 \text{ rad/s}$.

Velocidad en 1: $w = 1,77 \text{ rad/s}$

Velocidad en 2: $w = 2,09 \text{ rad/s}$

- Prueba 3: Velocidad estimada $w = 3,1 \text{ rad/s}$

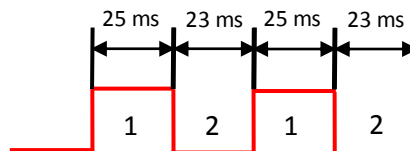


Figura 3.60.- Tiempo transcurrido entre flancos de la señal del sensor Hall en la tercera prueba, a $3,1 \text{ rad/s}$.

Velocidad en 1: $w = 2,76 \text{ rad/s}$

Velocidad en 2: $w = 3 \text{ rad/s}$

Se observó, de forma experimental, que la señal que generan los tres sensores Hall combinados no es perfectamente cuadrada, debido a que mecánicamente estos sensores no están posicionados en el ángulo preciso. En consecuencia, se podrían producir errores en la estimación de la velocidad angular.

Para solucionar este error, basta con medir el tiempo transcurrido sólo entre flancos de subida, o sólo entre flancos de bajada. De esta manera, la lectura de la velocidad será siempre la misma aunque la señal no sea perfectamente cuadrada. Como aspecto negativo de esta solución, la frecuencia con la que se recibirán las interrupciones externas será la mitad de la anterior. Ahora, la interrupción se disparará cuando se produzca un flanco descendente de la señal, como muestra la figura 3.61:

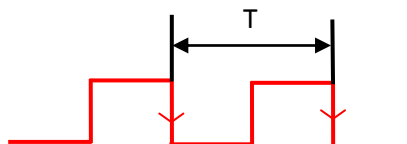


Figura 3.61.- Tiempo transcurrido T entre flancos descendentes de la señal de salida del circuito lógico.

La expresión (3.12) queda ahora de la siguiente manera:

$$f_{\text{pulsos}}(\text{Hz}) = \frac{\frac{37,5}{m} \cdot V(\text{m/s}) \cdot 3 \text{ sensores}}{2} = \frac{56,25}{m} \cdot V(\text{m/s}) \quad (3.14)$$

Y, en consecuencia, sólo se tendrán en cuenta los pulsos que ocurran cada 24° . La expresión (3.13) se modifica:

$$\omega = \frac{\frac{24^\circ}{T(s)}}{3 \text{ sensores}} \cdot \frac{\pi}{180^\circ} = \frac{0,140 \text{ rad}}{T(s)} \quad (3.15)$$

Sólo se realiza el cálculo la velocidad angular cuando se han obtenido ambos flancos descendentes de la señal ya que, si se realiza el cálculo a la frecuencia de la función *loop*, se podrían obtener valores erróneos. Para ello, se levanta una bandera cada vez que se obtiene el segundo flanco descendente, como se muestra en la figura 3.62:

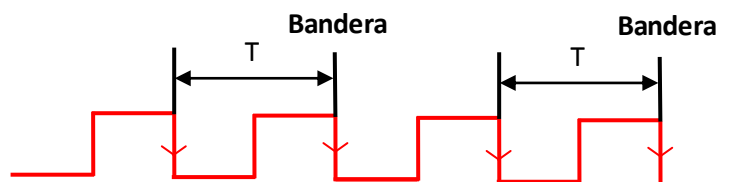


Figura 3.62.- Se activa una bandera cuando se tiene el tiempo transcurrido entre flancos descendentes. Esta bandera indica que ya se puede realizar el cálculo de la velocidad.

La figura 3.63 muestra la máquina de estados que maneja las interrupciones externas provenientes de la señal de los sensores Hall:

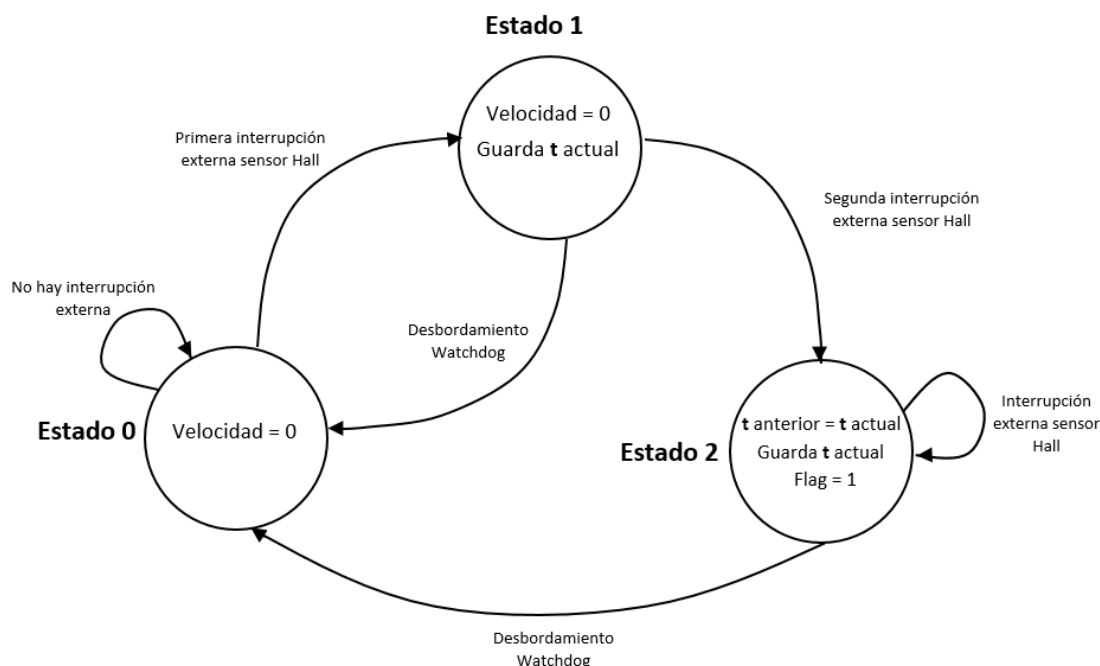


Figura 3.63.- Máquina de estados que representa el manejo de las interrupciones externas provenientes de los sensores Hall en el programa.

Se utiliza un *timer* como *watchdog* para detectar cuando la rueda se ha parado, o bien si se está moviendo a velocidad muy lentas (pocas interrupciones externas por segundo), para así poder saber cuándo hay que fijar el valor de la velocidad de las ruedas a 0.

El período de este *watchdog* es de 1 segundo, y se resetea cada vez que llega una interrupción externa proveniente del sensor Hall.

Estado 0: Vehículo parado o con velocidad muy lenta

Se llega a este estado si, por una parte, se ha iniciado el programa y por lo tanto es el primer estado de la secuencia, o bien porque se ha activado el *watchdog* mientras la secuencia se encuentra en los otros estados. El *watchdog* está ligado a una petición de interrupción interna, la cual llama a una ISR (Interrupt Service Routine) que fija la velocidad a 0 cuando el temporizador ha llegado a contar hasta 1 segundo sin que hayan habido interrupciones externas.

Estado 1: Primer flanco descendente de la señal

Se llega al estado 1 sólo cuando se ha obtenido el primer flanco descendente de la señal, el cual se tomará como referencia. Se guarda en una variable el instante de tiempo en el cual ha llegado este flanco. También se fuerza el valor de la velocidad angular a 0, para evitar que la variable de la velocidad guarde cualquier otro valor.

Estado 2: Cálculo de la velocidad angular

El programa llega al estado 2 si, después de haber obtenido el primer flanco descendente, llega una segunda interrupción externa de flanco descendente del Hall. Además, el programa se mantiene en este estado mientras el watchdog no se activa. Finalmente activa la bandera que permite realizar el cálculo de velocidad en la función *loop* del programa, ya que en este instante se puede obtener la diferencia de tiempo entre flancos, necesaria para realizar el cálculo de la velocidad angular.

3.5.2.3. Lectura de la dirección de la velocidad

Las señales de los sensores Hall se encuentran desfasadas entre ellas y, en consecuencia, se puede aprovechar esta característica para conocer la dirección del movimiento de la rueda.

La dirección de giro del motor se determinará comparando el nivel lógico de cada señal. Para ello, se considerará como referencia la señal del sensor Hall A. Si el motor gira en el sentido de las agujas del reloj (CW, *clockwise*), se observa como el estado del canal A es siempre inverso al del canal B, tal y como se representa en la figura 3.64:

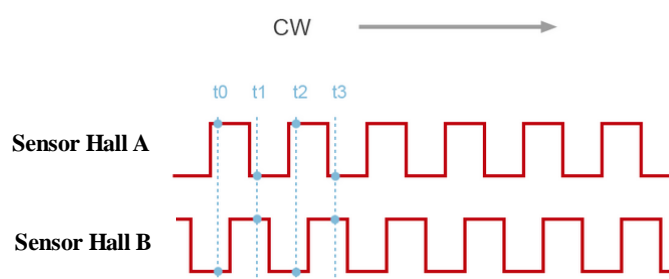


Figura 3.64.- Señales de los sensores Hall A y B al moverse en dirección a favor de las agujas del reloj.

Si se invierte el sentido de giro, contrario a las agujas del reloj (CCW, *counterclockwise*), e igualmente se toma como referencia la señal del sensor Hall A, se observa cómo tanto el estado de la señal del canal A como la de la señal B son siempre idénticas, tal y como se representa en la figura 3.65:

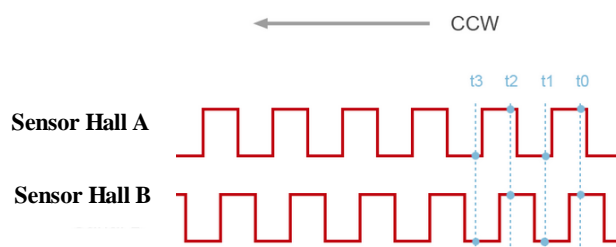


Figura 3.65.- Señales de los sensores Hall A y B al moverse en dirección contraria a las agujas del reloj.

En consecuencia, la lectura del sentido de giro del motor dependerá del canal que se tome como referencia. En cualquier caso, es posible diferenciar el sentido de giro simplemente comparando el nivel lógico de las señales obtenidas.

En el anexo [A2.11] se encuentra el código utilizado para obtener la dirección de movimiento de las ruedas. En la figura 3.66 se muestra el diagrama de flujo del programa:

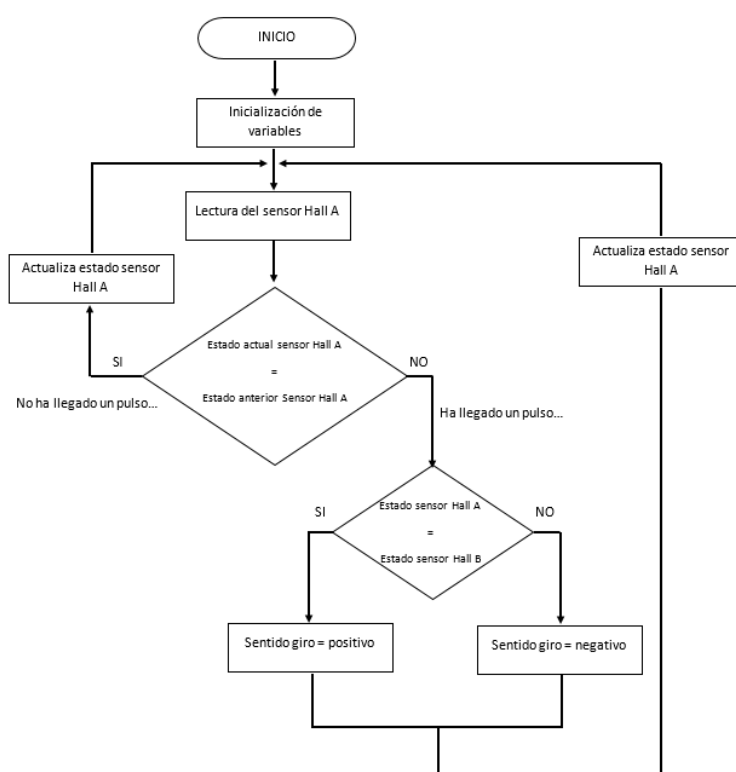


Figura 3.66.- Diagrama de flujo del algoritmo para detectar la dirección de giro de las ruedas.

4. Diseño e implementación de la solución

Como en la mayoría de proyectos de ingeniería, es recomendable llevar a cabo la creación de un primer prototipo, para así poder probar las funcionalidades del dispositivo diseñado (la silla de ruedas en este caso) de una manera más eficaz, y poder encontrar errores y deficiencias antes de crear el producto final.

El diseño de este prototipo de silla de ruedas de bajo coste está pensado para poder contener todos los elementos necesarios electrónicos y eléctricos que se han aprovechado del Hoverboard, en una estructura mecánica en la cual poder realizar pruebas de movimiento con peso humano. Los elementos que conformarán el prototipo de la silla de ruedas son los siguientes:

- Una estructura mecánica lo más parecida posible a una silla de ruedas convencional, como la mostrada en la figura 4.1, pero que además conecte todos los elementos electrónicos y eléctricos que necesita el sistema. Esta estructura debe de seguir la filosofía de bajo coste, por lo que tiene que ser sencilla, barata de construir, y debe permitir realizar pruebas de forma segura. En general, esta estructura deberá contener los siguientes elementos:
 - Cuatro ruedas: Dos motrices traseras y dos directrices delanteras (ruedas locas u orientables).
 - Una silla para realizar pruebas con carga humana.

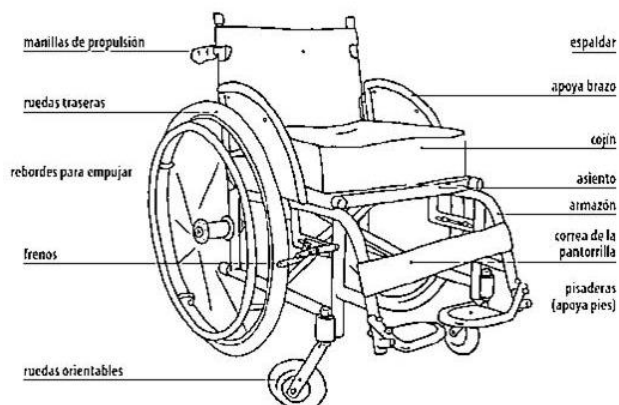


Figura 4.1.- Ejemplo de una estructura de silla de ruedas manual y sus posibles componentes mecánicos.

- Una etapa de potencia que proporcione la alimentación necesaria a todos los componentes electrónicos de la silla.
- Un sistema de control que permita controlar la velocidad de las ruedas a través de un joystick.

4.1. Estructura mecánica

Para construir una estructura prototipo en la cual realizar las pruebas necesarias de control de movimiento, se ha querido realizar algo sencillo pero robusto a la vez, teniendo en cuenta que esta estructura tiene que soportar el peso de una persona adulta, ya que uno de los objetivos del proyecto es realizar una silla de ruedas eléctrica adecuada a todos los públicos, no sólo para niños como es el caso del antecedente de este proyecto (Silla Exploradora Inteligente 1.0).

Durante el proyecto, se han construido dos modelos de estructuras diferentes, aunque ambos realizados con los mismos materiales, ya que en el primero de ellos se vieron algunas limitaciones.

En los siguientes apartados se explica brevemente en qué consistió este proceso de construcción.

4.1.1. Primer montaje

El primer montaje que se realizó constaba de una estructura hecha con perfiles de aluminio y guías (figura 4.2), y escuadras (figura 4.3), estas últimas con el objetivo de unir los perfiles.

La idea principal era montar una estructura en forma de "I" aprovechando el chasis del Hoverboard.



Figura 4.2.- Perfil de aluminio con una guía introducida en una de sus ranuras.



Figura 4.3.- Escuadras de metal para unir los perfiles de aluminio utilizados.

La figura 4.4 muestra la estructura en I realizada con los materiales mencionados anteriormente:

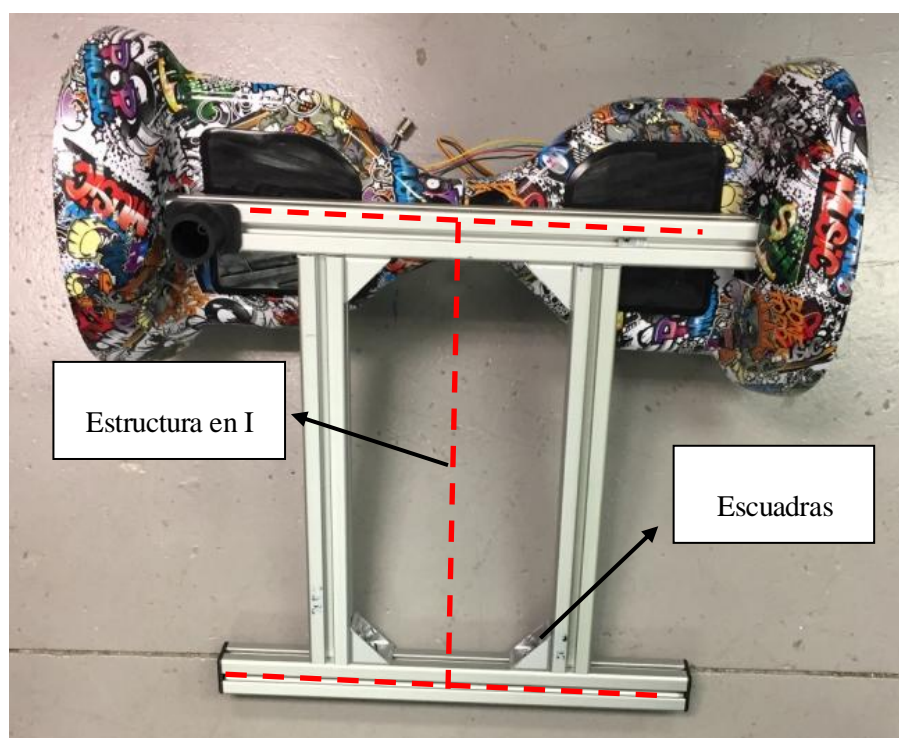


Figura 4.4.- Vista superior de la estructura en I realizada con perfiles de aluminio y escuadras.



Figura 4.5.- Vista lateral de la estructura en I realizada con perfiles de aluminio y escuadras.

Seguidamente, había que pensar qué tipo de asiento era el adecuado para que se pudieran realizar las pruebas de movimiento con peso. La solución más sencilla era utilizar una silla convencional, y hacer que sus patas estuvieran fijadas a la estructura en I apoyadas en impresiones 3D diseñadas específicamente para la silla. Estos soportes 3D se muestran en la figura 4.6:

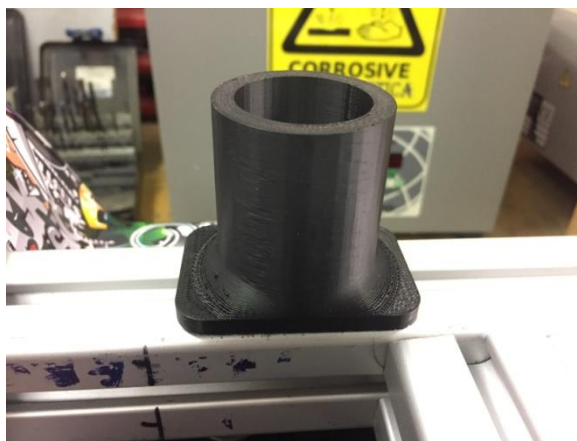


Figura 4.6.- Una de las impresiones 3D diseñadas para fijar las patas de la silla.

Finalmente, se añadieron dos ruedas locas las cuales facilitarían el movimiento de la estructura en general, tal y como se muestra en la figura 4.7:

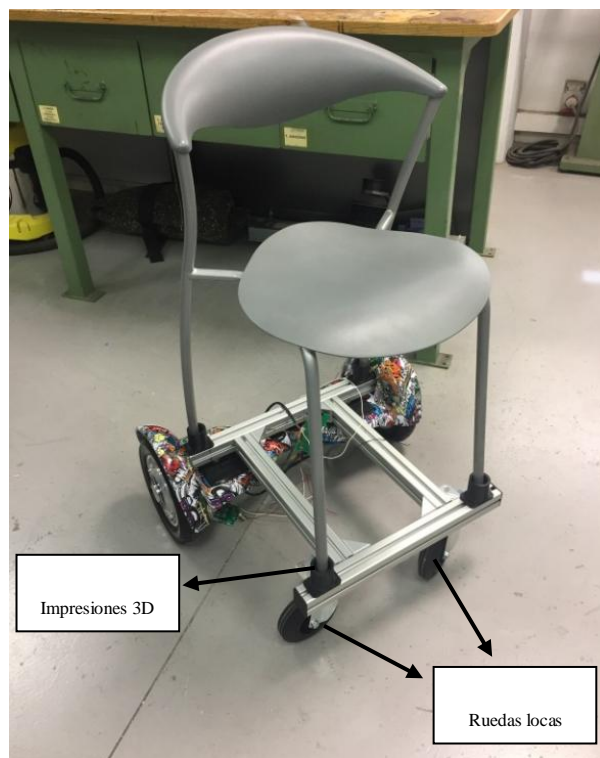


Figura 4.7.- Estructura final del primer montaje de la silla de ruedas eléctrica.

El principal problema que tenía esta estructura era que, por una parte, la electrónica del Hoverboard seguía estando incorporada dentro de su propio chasis, lo cual dificultaba su manejo para la realización de experimentos. Por otra parte, tampoco había ningún punto en la estructura donde fuese posible apoyar los pies de la persona, y eso era muy incómodo a la hora de realizar las pruebas de movimiento.

Con la intención de solucionar estos puntos se ideó una segunda estructura, explicada en el siguiente apartado.

4.1.2. Segundo montaje

En el segundo montaje se decidió construir una estructura rectangular, utilizando los mismos materiales que en el primero.

Asimismo, se decidió ampliar el tamaño de la estructura, añadiendo dos planchas de madera: una para contener los elementos electrónicos de la silla, y la otra para ser utilizada como reposapiés. También se extrajo el chasis de plástico del Hoverboard, y se colocaron sus elementos electrónicos en la plancha de madera mencionada anteriormente. El prototipo final de silla de ruedas realizado para el proyecto es el mostrado en la figura 4.8:

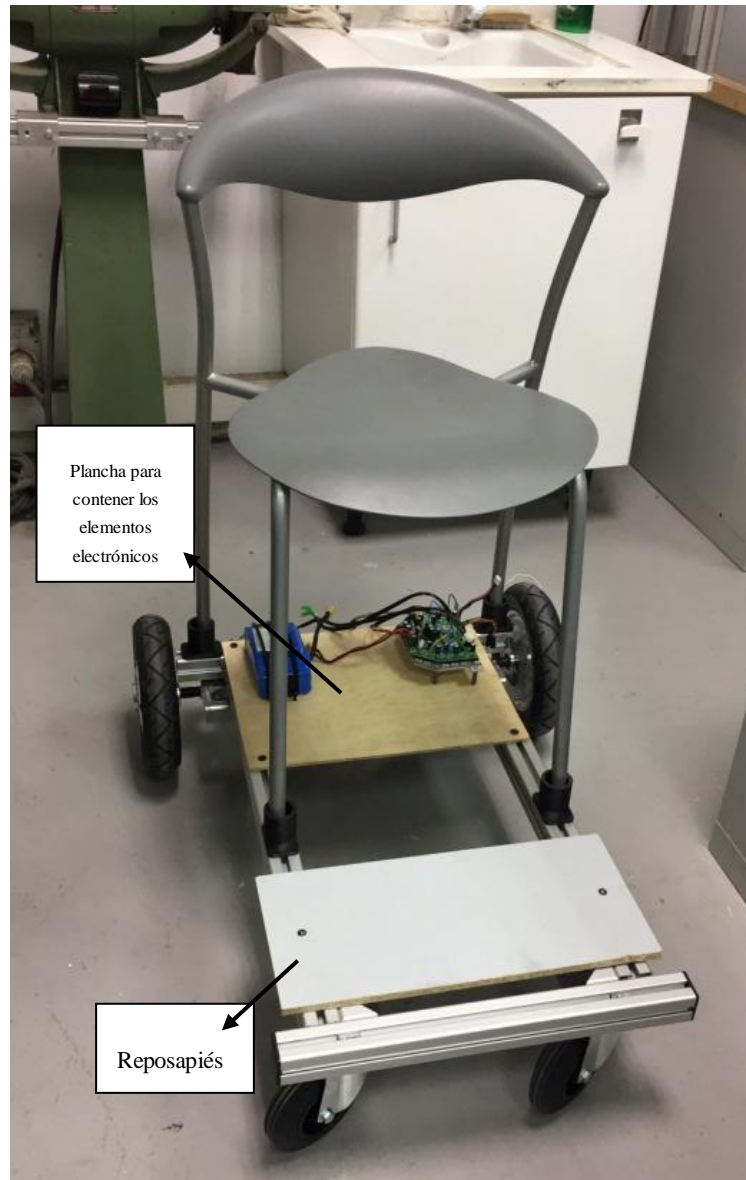


Figura 4.8.- Estructura final del primer montaje de la silla de ruedas eléctrica.

4.2. Etapa de potencia

Una vez realizada la estructura mecánica del prototipo de la silla de ruedas eléctrica, el siguiente paso fue diseñar una etapa de potencia para asegurar la correcta alimentación de todos los componentes electrónicos del sistema.

4.2.1. Circuito de potencia

El circuito de potencia del sistema se muestra en la figura 4.9:

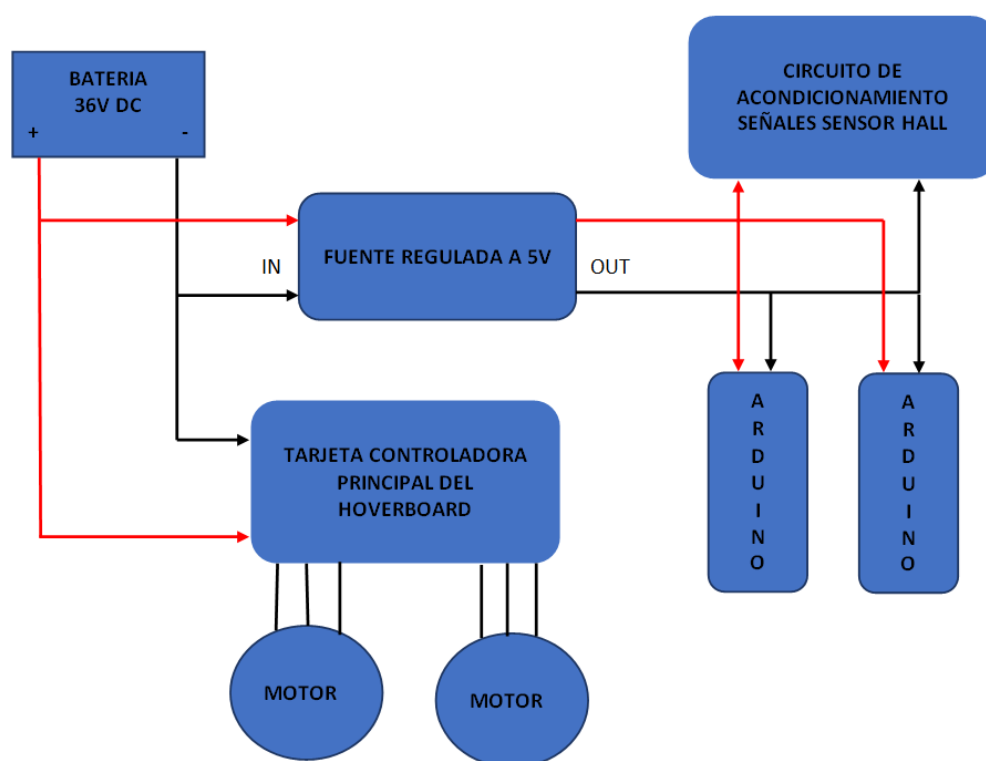


Figura 4.9.- Diagrama de la etapa de potencia que alimenta la electrónica de la silla de ruedas.

La batería de 36 Vdc aprovechada del Hoverboard alimenta a la controladora principal, la cual alimenta trifásicamente a los motores de ambas ruedas.

Sólo se ha tenido que adquirir una fuente regulada a 5 V que permite alimentar a 5 V la electrónica restante del sistema, como pueden ser los Arduino, o los circuitos electrónicos diseñados para procesar la señal de los sensores Hall.

4.3. Sistema de control

Una vez terminada la estructura mecánica de la silla y después de diseñar el circuito de potencia que alimentará a la electrónica del sistema, el siguiente objetivo fue implementar un control de velocidad por joystick para controlar la silla de ruedas.

Para ello, se añadirá la realimentación de velocidad de las ruedas que se ha conseguido obtener de los sensores Hall en el control de la plataforma.

4.3.1. Control con Arduino

La figura 4.10 muestra un diagrama de alto nivel de la implementación del control realizado con Arduino:

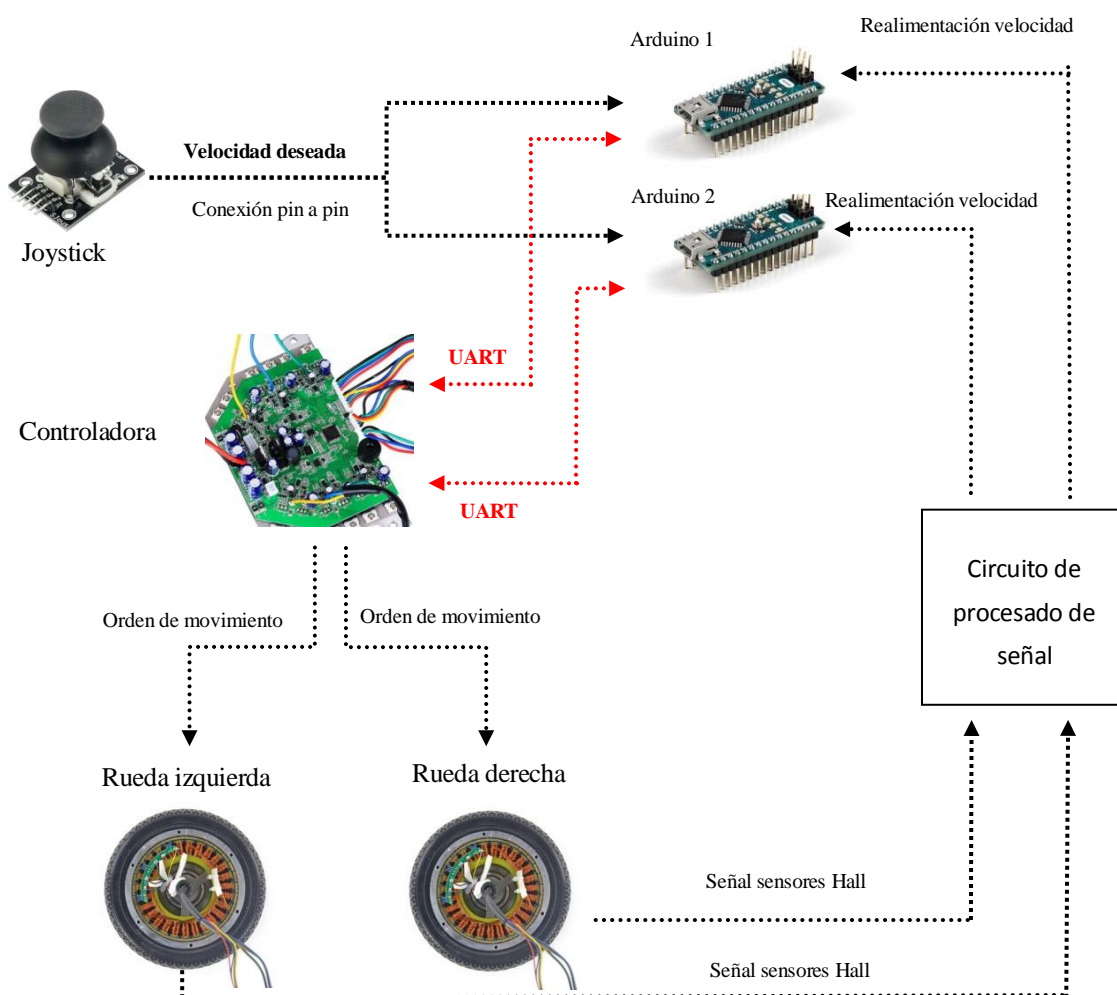


Figura 4.10.- Implementación del control de velocidad realizado con Arduino para la silla de ruedas.

4.3.1.1. Diseño del control PI

El control PI realizado en el proyecto tiene la siguiente estructura, mostrada en la figura 4.11:

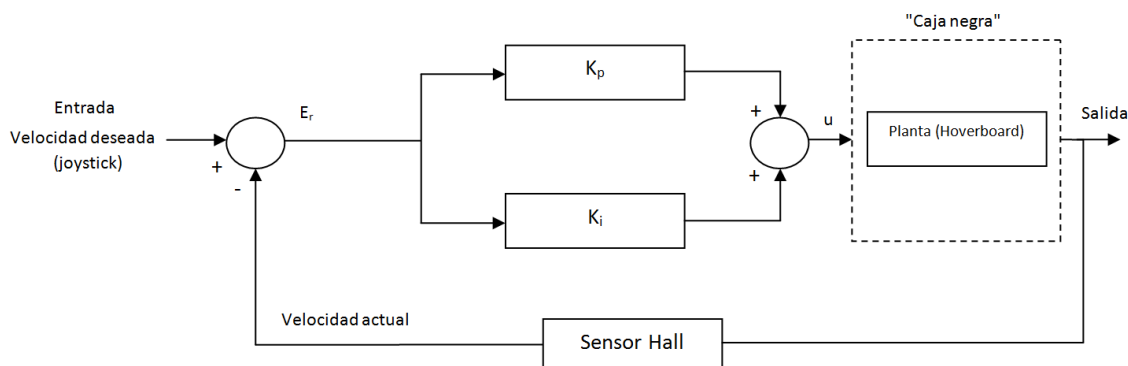


Figura 4.11.- Diagrama de bloques del control PI del proyecto.

En la figura 4.11 se ha dibujado la planta del Hoverboard como una caja negra, ya que es desconocida, y no se ha tenido el tiempo suficiente para diseñar un modelo robusto de la planta, debido a que presenta cierta complejidad por la no linealidad del sistema.

El primer bloque de control (bloque proporcional) consiste en el producto entre la señal de error E_r y la constante proporcional K_p . Con este bloque se pretende conseguir una respuesta rápida del sistema.

El segundo bloque de control (bloque integral) tiene como propósito disminuir y eliminar el error en estado estacionario. Este control integral actúa cuando hay una desviación entre la variable (velocidad actual) y el punto de consigna (velocidad mandada por joystick), integrando esta desviación en el tiempo y sumándola a la acción proporcional.

Del control mostrado en la figura 4.11 se extraen las siguientes expresiones:

El error entre la consigna (entrada) y velocidad actual (salida) se describe en la expresión 4.1:

$$E_r = w_d - w_a \quad (4.1)$$

Donde w_d es la velocidad deseada (consigna mandada por el joystick) y w_a , la velocidad actual leída de los sensores Hall.

Luego, el error integral E_i se ha definido como muestra la expresión 4.2:

$$E_i = E_r + E_i \quad (4.2)$$

La variable de control u se muestra en la expresión 4.3:

$$u = K_p \cdot E_r + K_i \cdot E_i \quad (4.3)$$

Esta variable de control u es la que recibirá la controladora del Hoverboard como consigna.

Respecto a la sintonización del controlador PI, existen diversos métodos para sintonizar las ganancias del controlador, ya sea por procedimientos teóricos como puede ser el posicionamiento de polos o diagrama de Bode, o por métodos experimentales como el método de Ziegler-Nichols.

Ninguno de los métodos comentados anteriormente eran posibles aplicarlos en el control del Hoverboard. Por una parte, los procedimientos teóricos como el posicionamiento de polos requieren conocer tanto la función de transferencia de la planta como sus parámetros, ambas cosas desconocidas.

Por otra parte, el método experimental de Ziegler-Nichols [28], que consiste en dos experimentos según la tipología de sistema, tampoco se pudo realizar. La sintonización del controlador por respuesta al escalón en la lazo abierto, como se muestra en la figura 4.12, es uno de ellos:

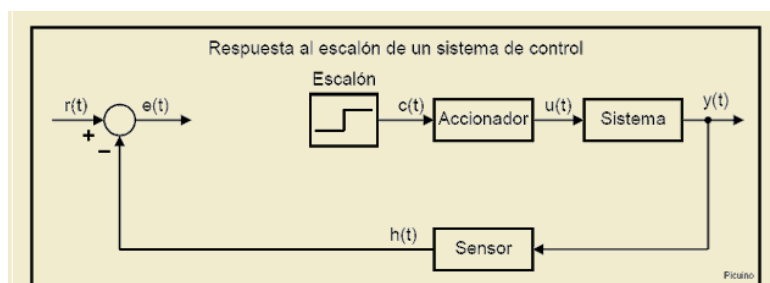


Figura 4.12.- Sistema en lazo abierto con entrada tipo escalón. [28]

Si se aplica una entrada de tipo escalón en lazo abierto al sistema del Hoverboard, éste se embalaría y provocaría una aceleración de la velocidad hasta llegar a la velocidad máxima, tal y como se habló en el apartado 3.4.2.

Luego, la sintonización del controlador en lazo cerrado siguiendo el método de Ziegler-Nichols tampoco se pudo realizar, dado que este método requiere que el sistema tenga una respuesta de carácter oscilante para poder obtener un valor de K_p crítica, con el cual se pueden obtener los demás parámetros.

Consecuentemente, la solución que se escogió fue realizar la sintonización del control experimentalmente.

Para ello, se ajustó primeramente el valor de la constante proporcional K_p hasta obtener una respuesta rápida del sistema. Luego, se ajustó K_i hasta reducir considerablemente el error de estado estacionario.

4.3.1.2. Software del control

El código del control PI se encuentra en los anexos [A2.12]. La mayoría del programa implementa algoritmos y funciones vistas anteriormente, en concreto, para el cálculo de la magnitud y dirección de la velocidad, el envío de la trama de datos necesaria para mover las ruedas, o el manejo de interrupciones externas provenientes del sensor Hall.

En la figura 4.13 se muestra el diagrama de flujo del programa en cuestión:

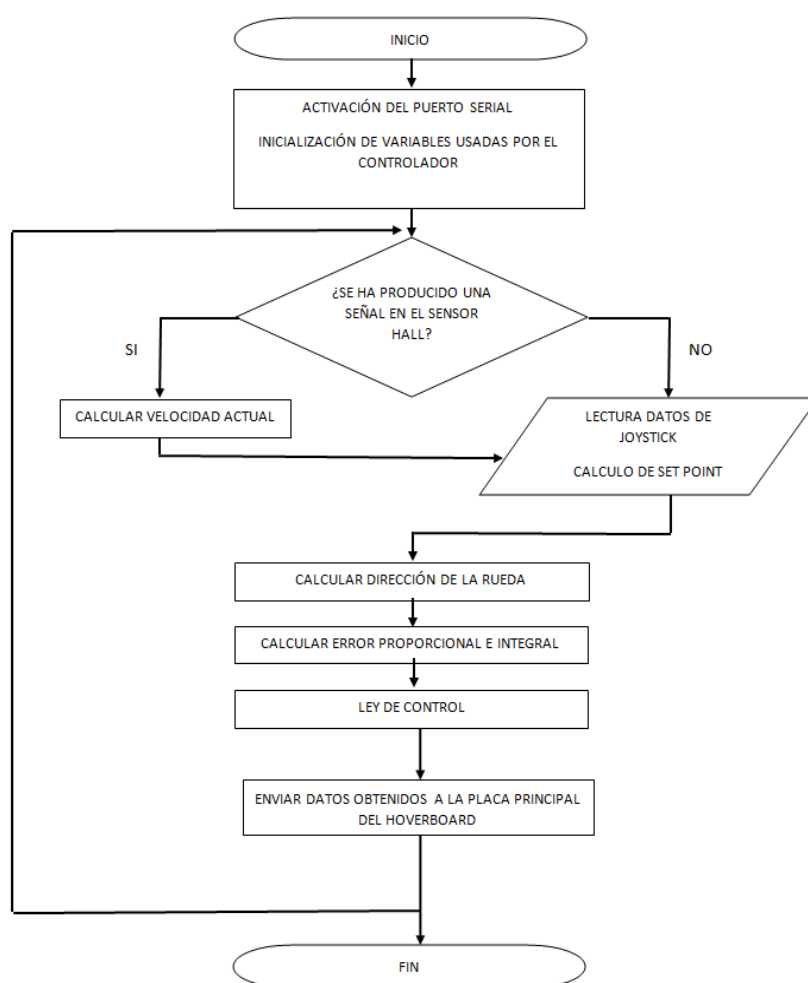


Figura 4.13.- Diagrama de flujo del control PI implementado.

4.3.1.3. Resultados experimentales

Se realizaron diversas pruebas a fin de encontrar qué valores de K_p y K_i eran los que hacían responder mejor al sistema. Se realizaron tres experimentos, cada uno en una condición diferente: en vacío sin peso, con la silla de ruedas en el suelo pero sin peso, y finalmente con el usuario montado en la silla de ruedas.

- Prueba en vacío:

Con la plataforma en vacío los valores de las constantes del control que hacían responder mejor al sistema eran $K_p = 0,2$ y $K_i = 0,4$. Si se elevaba mucho la constante proporcional K_p , la respuesta de las ruedas era más rápida, pero también más brusca. Si se aumentaba mucho el valor de K_i , el error de estado estacionario se corregía, pero también hacía que la respuesta del sistema se ralentizara considerablemente.

- Prueba con la silla de ruedas en el suelo:

En este experimento, las constantes con las que el sistema respondía adecuadamente eran las mismas que con las de la prueba en vacío.

- Prueba con el usuario montado:

Esta prueba con el usuario fue la más complicada para sintonizar el controlador. Para que la respuesta del sistema fuese la correcta, se tuvo que aumentar el valor de K_p , en concreto, hasta $K_p = 0,5$. Eso es debido a que el peso de la persona dificulta el movimiento de las ruedas, y si se mantenía el valor de K_p de los dos experimentos anteriores, las ruedas no tenían la suficiente fuerza para vencer el peso de la persona. Al aumentar el valor de K_p , ahora el sistema tenía una mayor fuerza al arrancar y, en consecuencia, podía vencer al peso de la persona.

Por otra parte, la constante integral K_i se disminuyó a $K_i = 0,2$. Se modificó este valor ya que el peso humano proporcionaba una inercia al sistema y, por ello, el sistema respondía lentamente a los cambios de consigna del joystick si el valor de K_i era demasiado alto.

Hay que resaltar que la sintonización del control PI realizado depende considerablemente del peso de la persona que está utilizando la silla de ruedas.

4.3.2. Control con Arduino y Raspberry Pi

El objetivo final que engloba el proyecto del Institut de Robòtica i Informàtica Industrial [1], tal y como se ha explicado al principio de la memoria, es construir una silla de ruedas eléctrica de bajo coste autónoma. Asimismo, en este proyecto se debe preparar el sistema de control para que, más adelante, se pueda implementar la navegación autónoma requerida. Para ello, el sistema debe de contener un ordenador embebido donde poder instalar ROS, con el cual se implementará el algoritmo de SLAM (Simultaneous Localization and Mapping). Es por ello que, en el control implementado anteriormente, se ha añadido una Raspberry Pi Model 3B [29], como ordenador embebido del sistema:

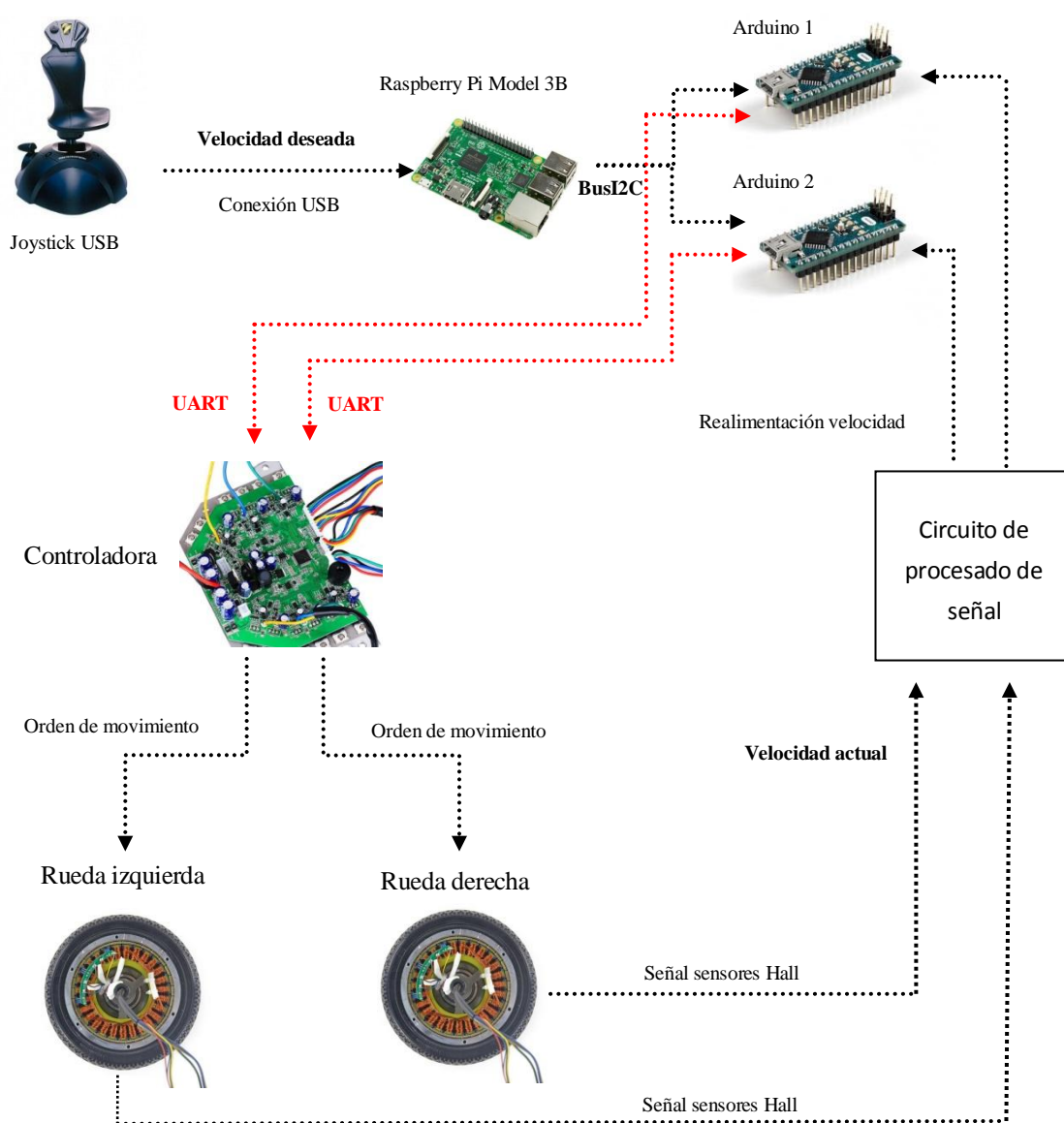


Figura 4.14.- Implementación del control de velocidad con Arduino y Raspberry Pi.

Como se puede observar en la figura 4.14, se ha cambiado el joystick de Arduino, el cual se conectaba directamente a las placas de Arduino pin a pin, por un joystick con comunicación USB, conectado ahora a la Raspberry Pi, con el objetivo de enviar las consignas a la silla de ruedas desde esta placa.

Además, para establecer una comunicación robusta y de poca complejidad entre Raspberry y Arduino, se decidió implementar un bus de comunicación I2C [30], ya que ambos dispositivos soportan este tipo de comunicación. En este caso, la Raspberry Pi es el master de la comunicación, mientras que los esclavos son las placas Arduino Nano.

4.3.2.1. Software del control

Para el manejo de este joystick, se han utilizado drivers y librerías creadas por el IRI, las cuales no se adjuntan en los anexos debido al gran tamaño del código.

Las placas de Arduino son las encargadas del control a bajo nivel del sistema (se encargan de leer la velocidad de las ruedas, y envían las tramas UART al Hoverboard). Por otro lado, la Raspberry Pi, a más alto nivel, es la encargada de leer las consignas del joystick para, seguidamente, enviárselas a los Arduinos.

Para que, tanto Arduino como Raspberry, puedan intercambiar las variables necesarias del control a través del bus I2C, se ha creado un mapa de registros, como el mostrado en la figura 4.15:

WHO_AM_I_ADDR	0x00
TARGET_SPEED_ADDR	0x01...0x04
CURRENT_SPEED_ADDR	0x09...0x0C
KP_ADDR	0x05...0x08
KI_ADDR	0x0D...0x10
KD_ADDR	0x11...0x14
I_MAX_ADDR	0x15...0x18

Figura 4.15.- Registros utilizados en el código.

A continuación, se explica brevemente en qué consiste cada registro:

WHO_AM_I_ADDR: Dirección de memoria donde se guarda la dirección de cada dispositivo de la comunicación I2C, de tipo *byte*.

TARGET_SPEED_ADDR: Dirección de memoria donde se guarda la consigna de velocidad del joystick, de tipo *float*.

CURRENT_SPEED_ADDR: Dirección de memoria donde se guarda la velocidad actual de las ruedas, de tipo *float*.

KP_ADDR: Dirección de memoria donde se guarda la constante proporcional K_p , de tipo *float*.

KI_ADDR: Dirección de memoria donde se guarda la constante proporcional K_i , de tipo *float*

KD_ADDR: Dirección de memoria donde se guarda la constante proporcional K_d , de tipo *float*, en el caso que se necesitare añadir en un futuro este parámetro.

I_MAX_ADDR: Dirección de memoria donde se guarda el valor límite del error integral, de tipo *float*.

El software diseñado se basa, en primer lugar, en un código [A3.1], implementado en ambas placas Arduino Nano. Este código realiza el algoritmo de control PI, la lectura de la velocidad de las ruedas y establece la comunicación UART con la controladora del Hoverboard.

Además, se han añadido las funciones necesarias para establecer una comunicación I2C con la Raspberry Pi, en concreto, utilizando funciones de la librería Wire de Arduino [31].

En segundo lugar, se ha creado una librería en C++ [A3.2] [A3.3], en la cual se han definido las variables y funciones necesarias para que todos los dispositivos puedan leer y escribir en el bus I2C.

Finalmente, con la API pública [A3.4] es posible utilizar todas las funciones de la librería, dependiendo de las necesidades específicas del usuario. Algunas de estas funciones son las siguientes:

- Set_speed_true: Función que permite establecer la velocidad angular de las ruedas.
- Get_speed_true: Función que permite obtener la velocidad actual de las ruedas.

- Set_control_true: Función que permite fijar las constantes K_p , K_i , y el límite del error integral del control PI.
- Get_control_true: Función que permite obtener las constantes K_p , K_i y el límite del error integral del control PI.

4.3.2.2. Resultados experimentales

Los valores de los parámetros del control PI, K_p y K_i , que hacen que el sistema responda correctamente, son los mismos que los encontrados en la sintonización con el control realizado en el apartado 4.3.1.3.

Además, se quisieron capturar diversas gráficas a fin de mostrar la respuesta del sistema en función de los distintos parámetros del control PI. No fue posible debido a que, si se querían capturar muchos datos por segundo, el bus I2C se saturaba. Eso es debido a que los Arduino Nano implementados en el control están trabajando al límite (interrupciones externas provenientes de los sensores Hall, escritura en el bus UART y escritura y lectura en el bus I2C).

5. Conclusiones y trabajo futuro

En este proyecto se ha descrito el diseño, la construcción, y el control de una silla de ruedas eléctrica de bajo coste, con el fin de que cualquier persona pueda reproducir este modelo para su uso. Asimismo, el sistema de control está preparado para que se pueda seguir desarrollando este proyecto para implementar la navegación autónoma en la silla de ruedas.

En primer lugar, se considera que durante el proyecto se ha seguido con éxito la filosofía de bajo coste, ya que los costes de los elementos utilizados para la creación del prototipo de la silla de ruedas han sido optimizados al máximo al haber aprovechado un producto ya comercializado, como es el caso del Hoverboard.

En segundo lugar, cabe señalar que ha sido un gran desafío realizar técnicas de ingeniería inversa en un producto el cual su funcionamiento se desconoce totalmente.

Por otro lado, se ha conseguido construir una estructura robusta, utilizando materiales baratos y sencillos de adquirir, que aguanta alrededor de los 100 kg, y que ha sido utilizada con éxito en las pruebas de movimiento y de control de velocidad.

Además, se ha conseguido implementar un control PI de velocidad el cual permite controlar el movimiento de la silla ruedas a través de un joystick y que, finalmente, tras haber verificado su funcionamiento, está listo para implementar la navegación autónoma.

En conclusión, la realización de este proyecto ha sido un gran desafío para mí, sobretodo el análisis de ingeniería inversa llevado a cabo, en el cual se han invertido muchas horas de trabajo. Todo ello, me ha hecho adquirir conocimientos muy importantes que me enriquecerán como persona y como profesional de la ingeniería.

Por otra parte, se han pensado diversas mejoras a implementar en el futuro en la silla de ruedas eléctrica de bajo coste.

En primer lugar, se quiere mejorar el control de velocidad, haciendo uso de un controlador más robusto. Para ello, se desea hacer una la modelización aproximada de la planta no lineal del Hoverboard, como se muestra en las figura 5.1 y 5.2:

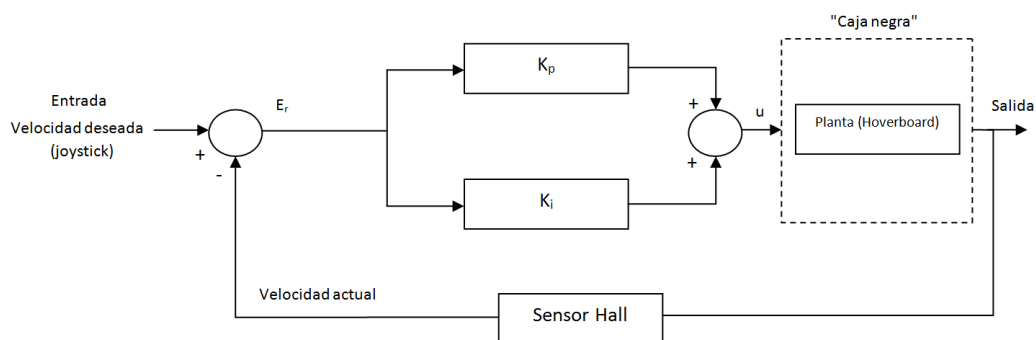


Figura 5.1.- Diagrama del control PI actual.

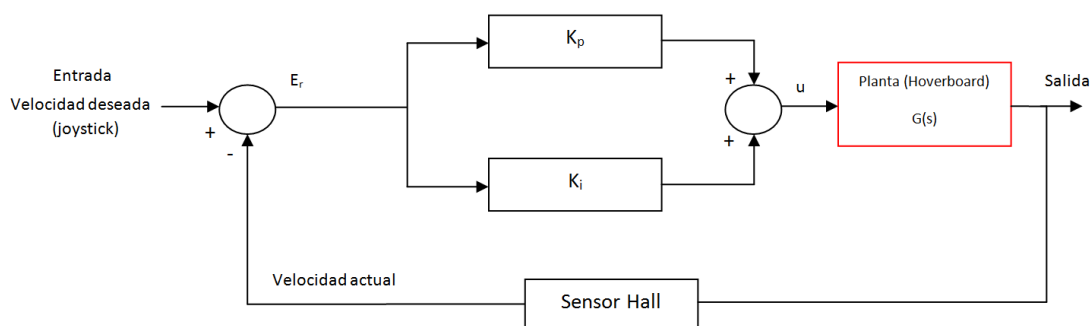


Figura 5.2.- Diagrama del control PI deseado en una futura mejora, donde se pretende conocer la función de transferencia del Hoverboard.

En segundo lugar, y como continuación a este proyecto, se quiere añadir la capacidad de localización y evasión de obstáculos a la silla de ruedas eléctrica. Para este punto, se pretende resolver el problema de SLAM (Simultaneous Localization and Mapping) en entornos interiores investigando distintos sensores de bajo coste, como por ejemplo una simple webcam, utilizando programario libre como es ROS, e implementando técnicas de construcción de mapas, localización y navegación.

Para realizar las mejoras planteadas anteriormente, se han propuesto las siguientes actividades con la finalidad de establecer las bases para la continuación del proyecto del IRI:

Sensorización

El objetivo de esta actividad será dotar a la plataforma de suficiente información sobre su entorno para poder navegar de forma segura y robusta, con un coste lo más reducido posible. Esta actividad se dividirá en los siguientes puntos:

- Evaluar el uso de sensores de ultrasonidos de bajo coste para poder detectar obstáculos en el camino de la silla de ruedas para así poder cambiar la trayectoria, reducir la velocidad o parar la silla de ruedas en el caso que sea necesario.
- Integrar los sensores de ultrasonidos a la plataforma móvil y al sistema de control. Desarrollar los drivers de ROS (Robot Operating System).
- Evaluar el uso del ratón óptico como encoder creado en este proyecto, con el objetivo de complementar la lectura de velocidad a través de los sensores Hall para obtener un sistema de odometría más robusto.
- Integrar el sistema de odometría a la plataforma móvil y al sistema de control. Desarrollo de los drivers de ROS.
- Evaluar el uso de cámaras web y de marcadores visuales repartidos por el entorno, localizados en posiciones conocidas, para poder realizar una localización global de la silla en el entorno donde se encuentra.
- Integrar estos sensores a la plataforma móvil y al sistema de control. Desarrollo de los drivers de ROS.
- Evaluar el uso de otros tipos de sensores para mejorar el comportamiento y la seguridad de la plataforma.

Sistema de navegación

El objetivo de esta actividad será desarrollar algoritmos de control y navegación que permitan mover y la plataforma y evitar obstáculos. La interfaz final para el usuario tiene que ser suficientemente simple para que la propia persona sea capaz de utilizarla. Esta actividad se desglosará en las siguientes actividades:

- Adaptar el sistema de navegación de ROS a las necesidades de la plataforma móvil, generando algoritmos de generación de trayectorias arbitrarias.
- Desarrollar una interfaz de usuario sencilla y de fácil acceso para personas con movilidad reducida.

Evaluación final de la plataforma

El objetivo de esta tarea será evaluar correctamente el funcionamiento de la plataforma completa para realizar la tarea deseada, colaborando estrechamente con la Fundación Nexe para poder adaptar la plataforma y la interfaz a las necesidades de los usuarios finales.

Paralelamente a los dos puntos anteriores, también se busca mejorar la estructura prototipo de la silla de ruedas, y llegar a un producto final. Con esto se hace referencia a crear una plataforma con un sistema de sedestación intercambiable ergonómico y atractivo visualmente, con la misma idea que la Silla Exploradora Inteligente 1.0 de Nexe Fundació.

Finalmente, como una de las premisas importantes del proyecto es que la silla sea de fácil reproducibilidad, se escribirá un manual o guía de ensamble que explique paso por paso el montaje mecánico y electrónico de la silla completa, con la finalidad de que cualquier persona pueda construir la silla de ruedas diseñada.

Presupuesto y análisis económico

En este apartado se incluye el presupuesto del proyecto, dividido en costes de material, costes de ingeniería y finalmente en el coste total del proyecto. Además, se ha considerado importante realizar un análisis económico del presupuesto, ya que uno de los objetivos más importantes era cumplir con la filosofía de bajo coste.

Costes de material

El coste del material utilizado para la realización de este proyecto se muestra en la tabla 6.1:

Tabla 6.1.- Costes del material del proyecto.

Concepto	Unidades	Precio unitario (€)	Total (€)
Hoverboard	1	200	200
Raspberry Pi	1	30	30
Arduino Nano	4	18	72
Joystick Arduino	1	5	5
Joystick USB	1	30	30
Electrónica General	1	30	30
Materiales estructura	1	80	80
TOTAL			447

447 € es el precio aproximado que tendría que invertir cualquier persona que quiera reproducir esta silla de ruedas. Es un precio más elevado que el de la Silla Exploradora Inteligente 1.0 de Nexe Fundació (300€), pero los componentes de la silla de este proyecto son mucho más robustos y, además, está pensada para todos los públicos (aproximadamente hasta 100 kg de peso).

Si se implementa un sistema de navegación autónoma, se tendrían que añadir entre dos y tres sensores de ultrasonidos [31] para la evasión de obstáculos, y un sensor Kinect [32] para la construcción del mapa del entorno por donde navega la silla. Orientativamente, el precio de

estos componentes es de aproximadamente 300 € . En total, el coste de material de la silla de ruedas autónoma ascendería los 750 €. Comparándolo con algunos diseños de silla autónomas que se han diseñado en los últimos años, (100.000 €) [33], (25.000 \$) [34], es un coste muy bajo que pondría una silla de ruedas eléctrica con navegación autónoma para entornos interiores al alcance de muchas personas con pocos recursos económicos.

Costes de ingeniería

En la tabla 6.2 se muestra los costes de ingeniería del proyecto, teniendo en cuenta un coste por hora de 25€/hora:

Tabla 6.2.- Costes de ingeniería del proyecto.

Concepto	Unidades (h)	Coste (€)
Formación conocimientos previos	50	1250
Proceso ingeniería inversa	400	10000
Diseño e implementación del control	250	6250
Montaje de la estructura	50	1250
Documentación	200	5000
TOTAL	950	23750

Coste total

Finalmente, el coste total del proyecto se muestra en la tabla 6.3:

Tabla 6.3.- Coste total del proyecto

Concepto	Unidades	Precio unitario (€)	Total (€)
Coste de material	1	447	447
Costes de ingeniería	1	23750	23750
Total			24197

Webgrafia

- [1] Institut de Robòtica i Informàtica Industrial. *Diseño y construcción de una silla exploradora inteligente de bajo coste* [en línia]. Disponible en: <https://www.iri.upc.edu/pfc/show/142>
- [2] Crmf Albacete. *Silla Exploradora Inteligente 1.0*. [en línia]. Disponible en: <http://www.crmfalbacete.org/recursosbajocoste/catalogo/Silla%20exploradora%20inteligente.pd>
- [3] Nexe Fundació. *Página principal de Nexe Fundació* [en línia]. Disponible en: <http://www.nexefundacio.org/es/>
- [4] Nexe Fundació. *Qué es la pluridiscapacidad* [en línia]. Disponible en: <http://www.nexefundacio.org/es/conoce-nexe/que-es-la-pluridiscapacidad/>
- [5] Nexe Fundació. *Bajo coste* [en línia]. Disponible en: <http://www.nexefundacio.org/es/bajo-coste/>
- [6] Nexe Fundació. *Xumet polsador web* [en línia]. Disponible en: <http://www.nexefundacio.org/es/cataleg/xumet-polsador-web/>
- [7] Youtube. *Silla Exploradora Inteligente versión 1.0* [en línia]. Disponible en: <https://www.youtube.com/watch?v=Dp9GmZk-tHE>
- [8] ROS. *Robot Operating System* [en línia]. Disponible en: www.ros.org
- [9] SLAM for Dummies. *A Tutorial Approach to Simultaneous Localization and Mapping* [en línia]. Disponible en: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslambblas_repo.pdf
- [10] Wired. *These Autonomous Wheelchairs Are the Future of Mobility* [en línia]. Disponible en: <https://www.wired.com/video/2017/02/autonomous-wheelchair/>
- [11] Hackaday. *Reverse Engineering Hoverboard Motor Drive* [en línia]. Disponible en: <https://hackaday.com/2016/06/10/reverse-engineering-hoverboard-motor-drive/>
- [12] FNAC. *Hoverboard COOLFUN Bluetooth 10"* [en línia]. Disponible en: <https://www.fnac.es/mp5580218/Hoverboard-COOLFUN-Bluetooth-10?oltype=1>
- [13] Cool and Fun. *Hoverboard User Manual* [en línia]. Disponible en: <https://www.hoverboard-coolfun.com/upload/download/user-manual-en.pdf>
- [14] Fictiv. *Hoverboard Teardown* [en línia]. Disponible en: <https://www.fictiv.com/blog/posts/hoverboard-teardown>

- [15] Drewspewmuse Blogspot. *How I hacked the Self Balancing Scooter* [en línea]. Disponible en: <http://drewspewmuse.blogspot.com.es/2016/06/how-i-hacked-self-balancing-scooter.html>
- [16] Sparkfun. *Serial Communication Tutorial* [en línea]. Disponible en: <https://learn.sparkfun.com/tutorials/serial-communication>
- [17] Arduino. *Software Serial Library* [en línea]. Disponible en: <https://www.arduino.cc/en/Reference/SoftwareSerial>
- [18] Addible GitHub. *9-bit serial output hack for Arduino*. [en línea]. Disponible en: <https://github.com/addibble/SoftwareSerial9>
- [19] Addible GitHub. *Hoverboard Controller*. [en línea]. Disponible en: <https://github.com/addibble/HoverboardController>
- [20] Control Tutorials for MATLAB and Simulink. *Inverted Pendulum: System Modeling* [en línea]. Disponible en: <http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum§ion=SystemModeling>
- [21] Ubuntu. *Directorios y sistemas de archivos* [en línea]. Disponible en: <https://help.ubuntu.com/kubuntu/desktopguide/es/directories-file-systems.html>
- [22] Kernel. *Input Protocol* [en línea]. Disponible en: <https://www.kernel.org/doc/Documentation/input/event-codes.txt>
- [23] Front and Back. *Qué son los niveles lógicos? Conectando Arduino y Raspberry* [en línea]. Disponible en: http://www.frontandback.org/laboratory/como_conectar_arduino_raspberry_pi
- [24] Digikey. *An Introduction to Brushless DC Motor Control* [en línea]. Disponible en: <https://www.digikey.com/en/articles/techzone/2013/mar/an-introduction-to-brushless-dc-motor-control>
- [25] Youtube. *Hoverboard Motor Schematic* [en línea]. Disponible en: <https://www.youtube.com/watch?v=HwKs2e75SQ>
- [26] GitHub. *The most affordable smooth and silent two wheels drive control* [en línea]. Disponible en: <https://github.com/OpHaCo/hoverbot>
- [27] MicroChip. *Rail-to-rail Input/Output, 10 Mhz Op Amps* [en línea]. Disponible en: <http://ww1.microchip.com/downloads/en/DeviceDoc/20001685E.pdf>

- [28] Picuino. *Método de Ziegler-Nichols* [en línea]. Disponible en: <https://sites.google.com/site/picuino/ziegler-nichols>
- [29] RS Components. *Raspberry Pi 3 Model B Datasheet* [en línea]. Disponible en: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf
- [30] Robots-argentina. *Comunicación Bus I2c. Descripción y funcionamiento* [en línea]. Disponible en: http://robots-argentina.com.ar/Comunicacion_busI2C.htm
- [31] Arduino. *Wire Library* [en línea]. Disponible en: <https://www.arduino.cc/en/Reference/Wire>
- [32] ElectroniLAB. *Sensor de distancia de Ultrasonido HC-SR05* [en línea]. Disponible en: <https://electronilab.co/tienda/sensor-de-distancia-de-ultrasonido-hc-sr05-hy-srf05/>
- [33] Robotics Business View. *3D Sensing: Kinect-ing Robots to their Environment* [en línea]. Disponible en: <https://www.generationrobots.com/en/401430-microsoft-kinect-sensor.html>
- [34] Universitat Ramon Llull. *La robótica mejora la movilidad de las personas con discapacidad intelectual usuarias de sillas de ruedas* [en línea]. Disponible en: <http://www.url.edu/es/sala-de-prensa/la-robotica-mejora-la-movilidad-de-las-personas-con-discapacidad-intelectual-usuarias-de-sillas-de-ruedas>
- [35] IEEE Spectrum. *Lidar-Equipped Autonomous Wheelchairs Roll Out in Singapore and Japan* [en línea]. Disponible en: <https://spectrum.ieee.org/transportation/self-driving/lidar-equipped-autonomous-wheelchairs-roll-out-in-singapore-and-japan>

Anexo A

A1. Primer proceso de ingeniería inversa

A1.1 Modificación del código de la referencia

```
1 #include <SoftwareSerial.h> //Librería SoftwareSerial modificada a 9 bits
2
3 #define MOSI 11 //MASTER OUTPUT SLAVE INPUT
4 #define MISO 12 //MASTER INPUT SLAVE OUTPUT
5 #define TX MOSI // TX = MOSI
6 #define RX MISO // RX = MISO
7 #define LEDPIN 13
8
9 SoftwareSerial mySerial(RX, TX);
10
11 void setup() {
12   mySerial.begin(31250); //La comunicación entre Arduino y Hoverboard es de 31250
baudios
13   Serial.begin(115200); //115200 baudios para monitor serial
14 }
15
16 char c = ' ';
17 signed int sp = 0;
18
19 void loop() {
20   /**Con el monitor serial de Arduino abierto***/
21   Serial.println(c);
22   if (c == ' ') {           // Si presionamos espacio, velocidad = 0;
23     sp = 0;
24   } else if (c == 'q') {    // Si presionamos q, reducimos en 10 la velocidad
25     sp -= 10;
26   } else if (c == 'w') {    //Si presionamos w, aumentamos en 10 la velocidad
27     sp += 10;
28   } else if (c == '2') {    //Si presionamos 2, aumentamos en 100 la velocidad
29     sp += 100;
30   } else if (c == '1') {    //Si presionamos 1, reducimos en 100 la velocidad
31     sp -= 100;
32   }
33   Serial.print("speed ");
34   Serial.println(sp);
35   Serial.print(" low byte ");
36   Serial.print((sp & 0xFF), HEX); //Se crea el byte bajo
```

```

37  Serial.print(" high byte ");
38  Serial.println((sp >> 8) & 0xFF, HEX); //Se crea el byte alto
39  do {
40      mySerial.write9(256);                //Inicio de trama
41      mySerial.write9(sp & 0xFF);           // Byte bajo de la "velocidad"
42      mySerial.write9((sp >> 8) & 0xFF);    //Byte alto de la "velocidad"
43      mySerial.write9(sp & 0xFF);           //Byte bajo de la "velocidad"
44      mySerial.write9((sp >> 8) & 0xFF);    //Byte alto de la "velocidad"
45      mySerial.write9(85);                  // 85 = Hay alguien montado, las
ruedas se pueden mover
46      mySerial.write9(88);                  //Número extra
47      mySerial.write9(88);                  //Número extra
48      delayMicroseconds(300);
49  } while (!Serial.available());
50  c = Serial.read();                        //Se lee el carácter introducido por el usuario
51  }

```

A2. Segundo proceso de ingeniería inversa

A2.1 *Wheelchair_sniffer.h*

```
1  #ifndef WHEELCHAIR_SNIFFER_H
2  #define WHEELCHAIR_SNIFFER_H
3
4  #include "threadserver.h"
5  #include "eventserver.h"
6  #include "commexceptions.h"
7  #include "eventexceptions.h"
8  #include "rs232.h"
9  #include <stdio.h>
10 #include <unistd.h>
11 #include <string>
12 #include <iostream>
13 #include "ctime.h"
14 #include <stdint.h>
15 #include <cstdlib>
16
17 #define BAUD_RATE 1000000 //460800
18 #define N_BITS 8
19 #define PARITY none
20 #define STOP_BITS 1
21
22 #define N_BYTES_COMMAND_SEND 11
23 #define N_BYTES_RESPONSE_SEND 3
24 #define MOTOR_HEADER_H 0xAA
25 #define MOTOR_HEADER_L 0x00
26 #define COMMAND_HEADER_H 0x01
27 #define COMMAND_HEADER_L 0x00
28 #define RESPONSE_HEADER_H 0x55
29 #define RESPONSE_HEADER_L 0x00
30
31 #define MOTOR_SYNC_BYTES 4
32 #define RESPONSE_SYNC_BYTES 2
33
34 #endif
```

A2.2 *Wheelchair_sniffer.cpp*

```

1  #include "wheelchair_sniffer.h"
2
3  int main(int argc, char *argv[])
4  {
5      if (argc != 3)
6      {
7          perror("It needs two arguments, the first is the motor serial port
8          and the second the response serial port.");
9          exit(EXIT_FAILURE);
10     }
11     CEventServer *event_server=CEventServer::instance();
12
13     CRS232 motor_sp("motor");
14     CRS232 response_sp("response");
15
16     TRS232_config serial_config;
17     serial_config.baud = BAUD_RATE;
18     serial_config.num_bits = N_BITS;
19     serial_config.parity = PARITY;
20     serial_config.stop_bits = STOP_BITS;
21
22     std::string motor_rx_event;
23     std::string response_rx_event;
24     std::list<std::string> events;
25
26     uint8_t motor_commands[N_BYTES_COMMAND_SEND];
27     uint8_t response_data[N_BYTES_RESPONSE_SEND];
28
29     CTime t;
30
31     long motor_time, response_time;
32
33     int event_id = 0;
34     uint8_t motor_index = 0, response_index = 0;
35     uint8_t motor_sync = 0, response_sync = 0;
36     uint8_t motor_sync_bytes[4] = {MOTOR_HEADER_H, MOTOR_HEADER_L,
37     COMMAND_HEADER_H, COMMAND_HEADER_L};
38     uint8_t response_sync_bytes[2] = {RESPONSE_HEADER_H, 39RESPONSE_HEADER_L};
39
40
41     try
42     {
43         motor_sp.open((void *)&argv[1]);
44         motor_sp.config(&serial_config);
45         motor_rx_event = motor_sp.get_rx_event_id();
46

```

```

47     response_sp.open((void *)&argv[2]);
48     response_sp.config(&serial_config);
49     response_rx_event = response_sp.get_rx_event_id();
50 }
51 catch(CCommException &e)
52 {
53     std::cout << e.what() << std::endl;
54     return 0;
55 }
56
57 events.push_back(motor_rx_event);
58 events.push_back(response_rx_event);
59 try
60 {
61     while (true)
62     {
63         event_id = event_server->wait_first(events,5000);
64         switch (event_id)
65         {
66             case 0://motor
67             {
68                 int num = motor_sp.get_num_data();
69                 unsigned char *bff;
70                 bff = new unsigned char[num];
71                 motor_sp.read(bff,num);
72                 int index = 0;
73                 while (index < num)
74                 {
75                     motor_commands[motor_index++] = bff[index++];
76                     switch (motor_sync)
77                     {
78                         case 0:
79                         case 1:
80                         case 2:
81                         case 3:
82                             if (motor_commands[motor_index-1] ==
83                                 motor_sync_bytes[motor_sync++])
84                             {
85                                 t.set();
86                                 motor_time = t.getTimeInMicroseconds();
87                             }
88                             else
89                             {
90                                 motor_sync = 0;
91                                 motor_index = 0;
92                             }
93                             break;
94                         case 4:
95                             if (motor_index == 1)//first data

```

```

96         {
97             t.set();
98             motor_time = t.getTimeInMicroseconds();
99         }
100     if (motor_index == N_BYTES_COMMAND_SEND)
101     {
102         std::cout << motor_time << ",";
103         for (uint8_t i=0; i<N_BYTES_COMMAND_SEND; i++)
104             std::cout << (int)motor_commands[i] << ",";
105         std::cout << std::endl;
106         motor_index = 0;
107     }
108     break;
109 }
110 }
111 delete[] bff;
112 }
113 break;
114 case 1://response
115 {
116     int num = response_sp.get_num_data();
117     unsigned char *bff;
118     bff = new unsigned char[num];
119     response_sp.read(bff,num);
120     int index = 0;
121     while (index < num)
122     {
123         response_data[response_index++] = bff[index++];
124         switch (response_sync)
125         {
126             case 0:
127             case 1:
128                 if (response_data[response_index-1] ==
129 response_sync_bytes[response_sync++])
130                 {
131                     t.set();
132                     response_time = t.getTimeInMicroseconds();
133                 }
134             else
135             {
136                 response_sync = 0;
137                 response_index = 0;
138             }
139             break;
140         case 2:
141             if (response_index == 1)//first data
142             {
143                 t.set();
144                 response_time = t.getTimeInMicroseconds();

```



```
145         }
146         if (response_index == N_BYTES_RESPONSE_SEND)
147         {
148             std::cout << response_time << ",";
149             for (uint8_t i=0; i<N_BYTES_RESPONSE_SEND; i++)
150                 std::cout << (int)response_data[i] << ",";
151             std::cout << std::endl;
152             response_index = 0;
153         }
154         break;
155     }
156 }
157 delete[] bff;
158 }
159 break;
160 }
161 }
162 }
163 catch(CCommException &e)
164 {
165     std::cout << e.what() << std::endl;
166     return 0;
167 }
168 catch(CEventTimeoutException &e)
169 {
170     std::cout << e.what() << std::endl;
171     return 0;
172 }
173 }
```

A2.3 Wheelchair Motor sniffer

```

1      #include <SoftwareSerial9.h>
2      #define SNIFFER_RX 13
3      #define SNIFFER_TX 12
4
5      #define COMMAND_PACKET_N 8
6      #define N_BYTES_COMMAND_SEND 11
7
8      #define MOTOR_HEADER_H 0xAA
9      #define MOTOR_HEADER_L 0x00
10
11     #define COMMAND_HEADER_H 0x01
12     #define COMMAND_HEADER_L 0x00
13     #define COMMAND_HEADER 0x0100
14
15     SoftwareSerial9 motor(SNIFFER_RX, SNIFFER_TX);
16
17     uint8_t motor_command[N_BYTES_COMMAND_SEND] = {MOTOR_HEADER_H, MOTOR_HEADER_L,
18     COMMAND_HEADER_H, COMMAND_HEADER_L, 0, 0, 0, 0, 0, 0};
19     uint8_t motor_index = N_BYTES_COMMAND_SEND - COMMAND_PACKET_N;
20
21     bool header_received = false;
22
23     void setup() {
24         // put your setup code here, to run once:
25         Serial.begin(1000000);
26         while (!Serial);
27
28         motor.begin(31250);
29         motor.listen(true);
30 }
31
32 void loop() {
33     // put your main code here, to run repeatedly:
34     uint16_t data_in;
35
36     if (motor.available())
37     {
38         data_in = motor.read();
39         if (!header_received)
40         {
41             if (data_in == COMMAND_HEADER)
42             {
43                 header_received = true;
44                 motor_index++;
45             }

```

```
46         }
47     else
48     {
49         motor_command[motor_index++] = data_in;
50     }
51 }
52 if (motor_index == N_BYTES_COMMAND_SEND)
53 {
54     /*for (uint8_t i=0; i<N_BYTES_COMMAND_SEND; i++)
55     {
56         Serial.print(motor_command[i],DEC);
57         Serial.print(",");
58     }
59     Serial.println(" ");*/
60     Serial.write(motor_command, N_BYTES_COMMAND_SEND);
61     motor_index = N_BYTES_COMMAND_SEND - COMMAND_PACKET_N;
62     header_received = false;
63 }
64 }
```

A2.4 Wheelchair Response Sniffer

```

1  #include <SoftwareSerial9.h>
2
3  #define SNIFFER_RX 13
4  #define SNIFFER_TX 12
5
6  #define RESPONSE_PACKET_N 1
7  #define N_BYTES_RESPONSE_SEND 3
8
9  #define RESPONSE_HEADER_H 0x55
10 #define RESPONSE_HEADER_L 0x00
11
12 SoftwareSerial9 response(SNIFFER_RX,SNIFFER_TX);
13
14 uint8_t response_data[N_BYTES_RESPONSE_SEND] = {RESPONSE_HEADER_H,
15 RESPONSE_HEADER_L, 0};
16 uint8_t resp_index = N_BYTES_RESPONSE_SEND - RESPONSE_PACKET_N;
17
18 void setup() {
19 // put your setup code here, to run once:
20 Serial.begin(921600);
21 while (!Serial);
22
23 response.begin(31250);
24 response.listen(true);
25 }
26
27 void loop() {
28 // put your main code here, to run repeatedly:
29 if (response.available ())
30 response_data[resp_index++] = response.read();
31
32 if (resp_index == N_BYTES_RESPONSE_SEND)
33 {
34     /*for (uint8_t i=0; i<N_BYTES_RESPONSE_SEND; i++)
35     {
36         Serial.print(response_data[i],DEC);
37         Serial.print(",");
38     }
39     Serial.println(" ");*/
40     Serial.write(response_data,N_BYTES_RESPONSE_SEND);
41     resp_index = N_BYTES_RESPONSE_SEND - RESPONSE_PACKET_N;
42 }
43 }
```

A2.5 Gráficos MATLAB

```

1      clc, clear all, close all
2      t = 0:1:12173;
3      h = zeros(12174,1);
4      %%%%%%%%%Placa Bateria; %%%%%%%%%
5      load bat_data_hs.m
6      load bat_data_ms.m
7      load bat_data_ls.m
8
9      a = bat_data_hs(:,1);
10     b = bat_data_hs(:,2);
11     c = bat_data_hs(:,5);
12     d = bat_data_hs(:,6);
13
14     lb = a(1:12174);
15     hb = b(1:12174);
16     l_n1 = c(1:12174);
17     acc = d(1:12174);
18
19     lb=uint16(lb);
20     hb=uint16(bitshift(hb,8));
21     u=bitor(lb,hb);
22     isNegative = int16(bitget(u,16));
23     convertedValue = int16(bitset(u,16,0)) + (-2^15)*isNegative;
24
25     figure(1)
26     subplot(3,1,1)
27
28     plot(t,convertedValue,'r',t,l_n1,'g',t,acc,'b',t, h,'k')
29     title('Batt Data High Speed'), xlabel('t'),ylabel('Data')
30     axis([0 12174 -500 500 ])
31
32     %%%%%%%%%
33     a = bat_data_ms(:,1);
34     b = bat_data_ms(:,2);
35     c = bat_data_ms(:,5);
36     d = bat_data_ms(:,6);
37
38     lb = a(1:12174);
39     hb = b(1:12174);
40     l_n1 = c(1:12174);
41     acc = d(1:12174);
42
43     lb=uint16(lb);
44     hb=uint16(bitshift(hb,8));
45     u=bitor( lb,hb);

```

```

46     isNegative = int16(bitget(u,16));
47     convertedValue = int16(bitset(u,16,0)) + (-2^15)*isNegative;
48
49     subplot(3,1,2)
50     plot(t,convertedValue,'r',t,l_n1,'g',t,acc,'b',t,h,'k')
51     title('Batt Data Medium Speed'), xlabel('t'),ylabel('Data')
52     axis([0 12174 -500 500 ])
53
54     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55     a = bat_data_ls(:,1);
56     b = bat_data_ls(:,2);
57     c = bat_data_ls(:,5);
58     d = bat_data_ls(:,6);
59
60     lb = a(1:12174);
61     hb = b(1:12174);
62     l_n1 = c(1:12174);
63     acc = d(1:12174);
64
65     lb=uint16(lb);
66     hb=uint16(bitshift(hb,8));
67     u=bitor( lb,hb);
68     isNegative = int16(bitget(u,16));
69     convertedValue = int16(bitset(u,16,0)) + (-2^15)*isNegative;
70
71     subplot(3,1,3)
72     plot(t,convertedValue,'r',t,l_n1,'g',t,acc,'b',t,h,'k')
73     title('Batt Data low Speed'), xlabel('t'),ylabel('Data')
74     axis([0 12174 -500 500 ])
75
76     pause()
77
78     %%%%%%%%%Placa PCB%%%%%%%%
79     t = 0:1:12793;
80     h = zeros(12794,1);
81
82     load pcb_data_hs.m
83     load pcb_data_ms.m
84     load pcb_data_ls.m
85
86     a = pcb_data_hs(:,1);
87     b = pcb_data_hs(:,2);
88     c = pcb_data_hs(:,5);
89     d = pcb_data_hs(:,6);
90
91     lb = a(1:12794);
92     hb = b(1:12794);
93     l_n1 = c(1:12794);
94     acc = d(1:12794);

```

```

95
96     lb=uint16(lb);
97     hb=uint16(bitshift(hb,8));
98     u=bitor( lb,hb);
99     isNegative = int16(bitget(u,16));
100    convertedValue = int16(bitset(u,16,0)) + (-2^15)*isNegative;
101
102    figure(2)
103    subplot(3,1,1)
104    axis([0 12794 -500 500 ])
105    plot(t,convertedValue,'r',t,l_nl,'g',t,acc,'b',t, h,'k')
106    title('PCB Data High Speed'), xlabel('t'),ylabel('Data')
107
108    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
109    a = pcb_data_ms(:,1);
110    b = pcb_data_ms(:,2);
111    c = pcb_data_ms(:,5);
112    d = pcb_data_ms(:,6);
113
114    lb = a(1:12794);
115    hb = b(1:12794);
116    l_nl = c(1:12794);
117    acc = d(1:12794);
118
119    lb=uint16(lb);
120    hb=uint16(bitshift(hb,8));
121    u=bitor( lb,hb);
122    isNegative = int16(bitget(u,16));
123    convertedValue = int16(bitset(u,16,0)) + (-2^15)*isNegative;
124
125    subplot(3,1,2)
126    axis([0 12794 -500 500 ])
127    plot(t,convertedValue,'r',t,l_nl,'g',t,acc,'b',t, h,'k')
128    title('PCB Data Medium Speed'), xlabel('t'),ylabel('Data')
129
130    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
131    a = pcb_data_ls(:,1);
132    b = pcb_data_ls(:,2);
133    c = pcb_data_ls(:,5);
134    d = pcb_data_ls(:,6);
135
136    lb = a(1:12794);
137    hb = b(1:12794);
138    l_nl = c(1:12794);
139    acc = d(1:12794);
140
141    lb=uint16(lb);
142    hb=uint16(bitshift(hb,8));
143    u=bitor( lb,hb);

```

```
144     isNegative = int16(bitget(u,16));
145     convertedValue = int16(bitset(u,16,0)) + (-2^15)*isNegative;
146
147     subplot(3,1,3)
148     axis([0 12794 -500 500 ])
149     plot(t,convertedValue,'r',t,l_nl,'g',t,acc,'b',t,h,'k')
150     title('PCB Data Low Speed'), xlabel('t'),ylabel('Data')
```

A2.6 Control aceleración Hoverboard

```

1      /*****/
2      #include <SoftwareSerial9.h>
3      #include <Arduino.h>
4      #define MOSI 11 //MASTER OUTPUT - SLAVE INPUT // TX MASTER
5      #define MISO 12 //MASTER INPUT - SLAVE OUTPUT // RX MASTER
6      #define TX MOSI
7      #define RX MISO
8
9      SoftwareSerial9 mySerial(RX,TX);
10
11     int Y = A0; //Potentiometer Y coordinate
12     int X = A1; //Potentiometer X coordinate
13
14     const int button = 2;
15
16     int LB = 0; //Low Byte
17     int HB = 0; //High Byte
18     volatile int l_n1 = 170; //ON-OFF Byte
19     volatile int flag = 1;
20
21     int Word = 0; //Low Byte + High Byte
22
23     int c = 0; //Angular velocity byte
24
25
26     void setup()
27     {
28         mySerial.begin(31250); //Hoverboard baudrate = 31250 baud
29         Serial.begin(115200); //Serial monitor baudrate = 115200 baud
30
31         attachInterrupt(digitalPinToInterrupt(button), interruptChange,
CHANGE);
32     }
33
34     void loop()
35     {
36
37
38     while((analogRead(Y)>512 || analogRead(Y)<512) && (analogRead(X)>=500 &&
analogRead(X)<=520))
39     {
40         Word = -(turn_order1(analogRead(Y))); // (+) Word for one wheel //
(-)Word for the other wheel (Data from each wheel behaves opposite)
41         LB = Word & 0xFF; //Low Byte
42         HB = (Word >> 8) & 0xFF; //High Byte

```

```

43     Hoverboard_data(LB,HB,l_nl); //Data sent to the controller
44     Serial.println(Word);
45     if((analogRead(Y)>=500 && analogRead(Y)<=520) && (analogRead(X)>=500
    && analogRead(X)<=520)) break;
46 }
47
48 while((analogRead(X)> 512 || analogRead(X)<512) && (analogRead(Y)>=500 &&
    analogRead(Y)<=520))
49 {
50     c = -(turn_order2(analogRead(X))); // (+) Word for one wheel // (-) Word
for the other wheel (Data from each wheel behaves opposite)
51     LB = c & 0xFF; //Low Byte
52     HB = (c >> 8) & 0xFF; //High Byte
53     Hoverboard_data(LB,HB,l_nl); //Data sent to the controller
54     Serial.println(Word);
55     if((analogRead(Y)>=500 && analogRead(Y)<=520) && (analogRead(X)>=500 &&
    analogRead(X)<=520)) break;
56 }
57
58}
59
60
61 void interruptChange() //Routine to change the ON-OFF byte with a button
62 {
63     if (digitalRead(button)== LOW) l_nl = 85;
64     else l_nl = 170;
65
66 }
67
68 void Hoverboard_data(int LB, int HB, int l_nl) //This function sends the
required data to control the hoverboard
69 {
70     mySerial.write9(256); //HEADER - Always 256
71     mySerial.write9(LB); //Low Byte
72     mySerial.write9(HB); //High Byte
73     mySerial.write9(LB); //Low Byte
74     mySerial.write9(HB); //High Byte
75     mySerial.write9(l_nl); //85 if anyone is riding the hoverboard
76     mySerial.write9(88); //Angular velocity byte
77     mySerial.write9(88); //Angular velocity byte
78 }
79
80 int turn_order1 (int value1) //This function does the mapping between the
range of values of the potentiometer and the values of minimum/maximum speed of
the hoverboard
81 {
82     int Word;
83
84     if(value1 >=400 || value1 <=600) Word = 0;

```

```
85   if(value1 <400) Word = map(analogRead(Y)-400,-400,-1,-200,0);
86   if(value1 >600) Word = map(analogRead(Y)-600,1,423,0,200);
87
88   return Word;
89 }
90
91 int turn_order2 (int value2) //This function does the mapping between the
range of values of the potentiometer and the values of minimum/maximum speed of
the hoverboard
92 {
93   int c;
94
95   if(value2 >=400 || value2 <=600) c = 0;
96   if(value2 <400) c = map(analogRead(X)-400,-400,-1,-250,0);
97   if(value2 >600) c = map(analogRead(X)-600,1,423,0,250);
98
99   return c;
100 }
```

A2.7 *mouse_encoder.h*

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <string>
5  #include <linux/input.h>
6  #include <fcntl.h>
7  #include <X11/Xlib.h>
8  #include <iostream>
9  #include <pthread.h>
10
11 #ifndef _MOUSE_ENCODER_H
12 #define _MOUSE_ENCODER_H
13
14 struct Mouse_odometry
15 {
16     double x_velocity;           //Velocity in X-axis
17     double y_velocity;           //Velocity in Y-axis
18     double x_coordinate;         //X coordinate
19     double y_coordinate;         //Y coordinate
20     double velocity;             //Velocity
21     double distancia;            //Total travelled distance
22 };
23
24 class CMouse_encoder
25 {
26     private:
27         const char* currentMouseDirectory;    //Directory where the mouse is
28         double currentConversionFactor;        //Mouse conversion factor
29         int fd;                                //File descriptor integer variable
30
31         pthread_t th1; //Thread creation
32         pthread_mutex_t mut; //Mutex creation
33
34         Mouse_odometry odom; //Struct object
35
36     protected:
37
38         static void* calculateOdometry(void *arg); //Function that computes the x,y
39         //velocity and gets the mouse coordinates
40
41     public:
42         CMouse_encoder(); //Default constructor
43         CMouse_encoder(const char*,double); //Constructor with arguments
44
45         ~CMouse_encoder(); //Default destructor

```

```
46
47
48     Mouse_odometry getOdometry(); //Function that gets the odometry parameters
49     void printOdometry();//Function that prints velocity and coordinates values
50
51     //Setters
52     void setDirectory(std::string mouseDirectory);
53     void setConversionFactor(float conversionFactor);
54
55     //Getters
56     std::string getDirectory();
57     float getConversionFactor();
58 };
59 #endif
```

A2.8 *mouse_encoder.cpp*

```
1  #include "mouse_encoder.h"
2
3  CMouse_encoder::CMouse_encoder(const char* mouseDirectory,double
conversionFactor) //Constructor with parameters ( mouse directory and conversion
factor)
4  {
5      currentMouseDirectory = mouseDirectory;
6
7      currentConversionFactor = conversionFactor;
8
9      if((fd = open(CMouse_encoder::currentMouseDirectory, O_RDONLY)) == -1) //We
open the mouse device, read only
10     {
11         perror("Opening Failure");
12         exit(EXIT_FAILURE);
13     }
14
15     odom.x_coordinate = 0;//We assure the coordinate X is always initialized at 0
16     odom.y_coordinate = 0;//We assure the coordinate Y is always initialized at 0
17
18     int res;
19
20     res = pthread_mutex_init(&(this->mut), NULL); //Mutex initialization
21     if (res != 0)
22     {
23         perror("Mutex initialization failed");
24         exit(EXIT_FAILURE);
25     }
26
```

```

27     res= pthread_create(&th1, NULL,calculateOdometry, (void *)this); //Thread
initialization
28     if (res != 0)
29     {
30         perror("Thread initialization failed");
31         exit(EXIT_FAILURE);
32     }
33
34 }
35
36 CMouse_encoder::~CMouse_encoder() //Destructor
37 {
38     pthread_mutex_destroy(&(this->mut));
39     pthread_join(th1,NULL);
40 }
41
42 void *CMouse_encoder::calculateOdometry(void *arg)
43 {
44     struct input_event ie; //Defined in /linux/input.h.
45
46     long newT = 0;          //Current time of the event
47     long oldT = 0;          //Previous time before the event
48     long diffT = 0;         //Elapsed time
49
50     double aux_X;           //Auxiliar X coordinate (debugging purpose)
51     double aux_Y;           //Auxiliar Y coordinate (debugging purpose)
52
53
54     CMouse_encoder *enc = (CMouse_encoder *) arg;
55
56     while(read(enc->fd, &ie, sizeof(struct input_event)))//While mouse events are
happening...
57     {
58
59         pthread_mutex_lock(&(enc->mut));
60
61         if (ie.type == EV_REL) //If detected EV_REL event type (relative movement)
62         {
63             if (ie.code == REL_X)//If X movement detected (REL_X == 0)
64             {
65                 enc->odom.x_coordinate += ie.value*enc->currentConversionFactor; //X
coordinate updated
66                 aux_X = ie.value;
67             }
68
69             else if (ie.code == REL_Y) //If Y movement detected (REL_Y == 1)
70             {
71                 enc->odom.y_coordinate += ie.value*enc->currentConversionFactor; //Y
coordinate updated

```

```

72         aux_Y = ie.value;
73     }
74
75     //Updating the time between events
76
77     oldT = newT;
78     newT = ie.time.tv_usec;
79
80
81     //Time offset control, ie.time.tv_usec goes from 0 to 1 milion us.
82     if(newT < oldT)
83     {
84         long auxT = newT + 1000000;
85         diffT = (long)auxT-oldT;
86     }
87
88     else
89     {
90         diffT = (long)newT-oldT;
91     }
92
93     //Computations of total distance and velocity in X and Y axis.
94
95         enc->odom.distancia+=(abs(ie.value))*enc->currentConversionFactor;
//Total distance (accummulated).
96         enc->odom.x_velocity = (aux_X*enc->currentConversionFactor)/(diffT*1E-6);
//Velocity in X-axis.
97         enc->odom.y_velocity = (aux_Y*enc->currentConversionFactor)/(diffT*1E-6);
98//Velocity in Y-axis.
99     }
100
101     pthread_mutex_unlock(&(enc->mut));
102
103 }
104
105     return NULL;
106 }
107
108
109
110 Mouse_odometry CMouse_encoder::getOdometry() //Function that returns the
struct of the odometry
111 {
112     pthread_mutex_lock(&(this->mut));
113
114     Mouse_odometry odom = {this->odom.x_velocity,this->odom.y_velocity,this-
>odom.x_coordinate,this->odom.y_coordinate};
115
116     pthread_mutex_unlock(&(this->mut));
117
118     return odom;

```

```

119     }
120
121 void CMouse_encoder::printOdometry() //Function that prints the odometry
122 {
123     pthread_mutex_lock(&(this->mut));
124
125     std::cout<<"Total traveled distance: "<<odom.distancia<<std::endl;
126
127     std::cout<<"X-Coordinate: "<<odom.x_coordinate<<std::endl;
128     std::cout<<"Y-Coordinate: "<<odom.y_coordinate<<std::endl;
129     if(!(odom.x_velocity>2 || odom.y_velocity>2) )
130     {
131         std::cout<<"X-Velocity: "<<odom.x_velocity<<std::endl;
132         std::cout<<"Y-Velocity: "<<odom.y_velocity<<std::endl;
133     }
134
135     pthread_mutex_unlock(&(this->mut));
136 }

```

A2.9 *mouse_encoder_test.cpp*

```

1  #include "mouse_encoder.h"
2  #include <iostream>
3
4  int main(int argc, char *argv[])
5  {
6      //double timeout = 45.0;
7      //double t = 0.0;
8      std::cout << "-----" << std::endl;
9      std::cout << "Mouse Encoder Example" << std::endl;
10     std::cout << "-----" << std::endl;
11
12     //CMouse_encoder mouse2("/dev/input/event10",53E-6); //Mouse no.2
13     CMouse_encoder mouse1("/dev/input/event11",53E-6); //Mouse no.1
14
15     mouse1.getOdometry();
16     //mouse2.getOdometry();
17
18     while(1)
19     {
20         mouse1.printOdometry();
21         //mouse2.printOdometry();
22         usleep(10000);
23     }
24 }

```


A2.10 Lectura de la magnitud de la velocidad

```

1      /*****HOVERBOARD DRIVER*****/
2      #include <SoftwareSerial9.h>
3      #include <Arduino.h>
4      #include "Timer1.h"
5
6      #define MOSI 11 //MASTER OUTPUT - SLAVE INPUT // TX MASTER
7      #define MISO 12 //MASTER INPUT - SLAVE OUTPUT // RX MASTER
8      #define TX MOSI
9      #define RX MISO
10
11     #define outputA 5 //A Hall sensor signal
12     #define outputB 6 //B Hall sensor signal
13
14     SoftwareSerial9 mySerial (RX,TX);
15
16     /*****Wheel speed*****/
17     volatile unsigned long tiempo; //Time elapsed between Hall edges
18     volatile float w=0.0 //Angular velocity
19     volatile unsigned long newT = 0;
20     volatile unsigned long oldT = 0;
21     /*****/
22
23     /*****Flags*****/
24     volatile int state = 0;
25     volatile int flag1 = 0;
26     /*****/
27
28     /*****Interrupts*****/
29     const int button = 2;
30     const int hall_sensor = 3;
31 /*****/
32     void setup()
33     {
34         mySerial.begin(31250); //Hoverboard baudrate = 31250 baud
35         //Serial.begin(115200); //Serial monitor baudrate = 115200 baud
36
37         attachInterrupt (digitalPinToInterrupt (button), interruptChange,
CHANGE);
38         attachInterrupt (digitalPinToInterrupt (hall_sensor), interruptHall,
FALLING);
39
40         pinMode (outputA, INPUT); //Hall A

```

```

41     pinMode (outputB, INPUT); //Hall B
42     startTimer1(1500000);      //Timer overflows at 1,5 seconds
43
44     aLastState = digitalRead(outputA); //Reading Hall A
45
46     state = 0;
47     w = 0; //Initializing velocity to 0
48     newT = millis();
49     oldT = 0;
50 }
51
52 void loop()
53 {
54     /*****CURRENT SPEED CALCULATION*****/
55     if(flag1 == 1) //A hall signal
56     {
57         tiempo = newT-oldT;
58         w =(8000.0*3.14*100.0)/((float)tiempo);
59         flag1 = 0;
60     }
61 }
62 ISR(timer1Event) //Watchdog routine
63 {
64     state = 0;
65     w = 0;
66 }
67
68 void interruptHall() //Interrupt function for the Hall sensor
69 {
70     resetTimer1();
71
72     if(state == 0)
73     {
74         state = 1;
75         w = 0;
76         newT = millis();
77     }
78     else if(state == 1 || state == 2)
79     {
80         state = 2;
81         flag1 = 1;
82         oldT = newT;
83         newT = millis();
84     }
85 }

```

A2.11 Lectura de la dirección de la velocidad

```

1      /*****HOVERBOARD DRIVER*****/
2      #include <SoftwareSerial9.h>
3      #include <Arduino.h>
4
5      #define MOSI 11 //MASTER OUTPUT - SLAVE INPUT // TX MASTER
6      #define MISO 12 //MASTER INPUT - SLAVE OUTPUT // RX MASTER
7      #define TX MOSI
8      #define RX MISO
9
10     #define outputA 5 //A Hall sensor signal
11     #define outputB 6 //B Hall sensor signal
12
13     SoftwareSerial9 mySerial(RX,TX);
14
15     void setup()
16     {
17         mySerial.begin(31250); //Hoverboard baudrate = 31250 baud
18         //Serial.begin(115200); //Serial monitor baudrate = 115200 baud
19
20         attachInterrupt(digitalPinToInterrupt(button),interruptChange,
CHANGE);
21         attachInterrupt(digitalPinToInterrupt(hall_sensor),interruptHall,
FALLING);
22
23         pinMode (outputA,INPUT); //Hall A
24         pinMode (outputB,INPUT); //Hall B
25
26         aLastState = digitalRead(outputA); //Reading Hall A
27     }
28
29     void loop()
30     {
31         Serial.println(wheelDirection);
32     }
33
34     void wheelDirectionx()
35     {
36
37         aState = digitalRead(outputA); // Reads the "current" state of the outputA
38         // If the previous and the current state of the outputA are different, that
means a Pulse has occurred
39         if (aState != aLastState)
40         {
41             // If the outputB state is different to the outputA state, that means
the encoder is rotating clockwise

```

```
42     if (digitalRead(outputB) != aState)
43     {
44         wheelDirection = 1;
45     }
46     else
47     {
48         wheelDirection = -1;
49     }
50
51 }
52 aLastState = aState; // Updates the previous state of the outputA with the
current state
53}
```

A2.12 Control de velocidad con Arduino

```

/*****HOVERBOARD DRIVER*****/
1    #include <SoftwareSerial9.h>
2    #include <Arduino.h>
3    #include "Timer1.h"
4
5    #define MOSI 11 //MASTER OUTPUT - SLAVE INPUT // TX MASTER
6    #define MISO 12 //MASTER INPUT - SLAVE OUTPUT // RX MASTER
7    #define TX MOSI
8    #define RX MISO
9
10   #define outputA 5 //A Hall sensor signal
11   #define outputB 6 //B Hall sensor signal
12
13   SoftwareSerial9 mySerial (RX,TX);
14
15   /*****Joystick*****/
16   int Y = A0; //Potentiometer Y coordinate
17   int joystick = 512;
18   float setpoint = 0.0;
19   /*****/
20
21
22   /*****Hoverboard Data*****/
23   int Word = 0; //Low Byte + High Byte
24   int LB = 0; //Low Byte
25   int HB = 0; //High Byte
26   volatile int l_n1 = 170;
27   int con;
28   /*****/
29
30   /****Control P****/
31   float errorP = 0.0;
32   float Kp = 0.3;
33   /*****/
34
35   /****Control I****/
36   float errorI = 0.0;
37   float Ki = 0.1;
38   float dt = 0.01;
39   /*****/
40
41   /*****Wheel speed*****/
42   volatile unsigned long tiempo;
43   volatile float w=0.0;
44   volatile unsigned long newT = 0;
45   volatile unsigned long oldT = 0;

```

```

46  /*****/
47
48  /*****Flags*****/
49  volatile int state = 0;
50  volatile int flag1 = 0;
51  /*****/
52  int contador = 0;
53  /**Dirección Rueda**/
54  int aState;
55  int aLastState;
56  int wheelDirection;
57  /*****/
58
59  /*****Interrupts*****/
60  const int button = 2;
61  const int hall_sensor = 3;
62  /*****/
63  void setup()
64  {
65      mySerial.begin(31250); //Hoverboard baudrate = 31250 baud
66      //Serial.begin(115200); //Serial monitor baudrate = 115200 baud
67
68      attachInterrupt(digitalPinToInterrupt(button), interruptChange,
69      CHANGE);
70      attachInterrupt(digitalPinToInterrupt(hall_sensor), interruptHall,
71      FALLING);
72
73      pinMode (outputA, INPUT); //Hall A
74      pinMode (outputB, INPUT); //Hall B
75
76      startTimer1(1500000); //Timer overflows at 1,5 seconds
77
78      aLastState = digitalRead(outputA); //Reading Hall A
79
80      state = 0;
81      w = 0;
82      newT = millis();
83      oldT = 0;
84  }
85
86  void loop()
87  {
88      //contador++;
89      /*****CURRENT SPEED CALCULATION*****/
90      if(flag1 == 1) //A hall signal
91      {
92          tiempo = newT-oldT;
93          w = (8000.0*3.14*100.0)/((float)tiempo);
94          flag1 = 0;

```

```

93         }
94     /*****/
95
96     /*****READING OF THE DESIRED SPEED (Joystick Input)*****/
97     joystick = analogRead(Y);      // Analog read of the joystick
98     setpoint = desired_w(joystick); //Desired speed
99     wheelDirectionx();              //Direction of the wheel (1 or -1)
100    /*****/
101    //if(contador == 50)
102    //{
103    //contador = 0;
104
105    /*****PI CONTROL*****/
106    errorP = setpoint - w*((float)wheelDirection)/180.0; // Proportional error
107    errorI = errorI + errorP*dt;                          // Integral error
108
109    if (errorI >=250) //Saturation of the integral error
110    {
111        errorI = 250;
112    }
113    else if(errorI <=-250)
114    {
115        errorI = -250;
116    }
117    /*****/
118    //}
119    /*****CONTROL VARIABLE*****/
120    Word = (Kp*errorP)+(Ki*errorI);
121    /*****/
122    /***DATA SENDED TO THE HOVERBOARD***
123    LB = Word & 0xFF; //Low Byte
124    HB = (Word >> 8) & 0xFF; //High Byte
125    Hoverboard_data(LB,HB,l_nl);
126    /*****/
127    //Serial.println(contador);
128    }
129
130
131    ISR(timer1Event) //Watchdog routine
132    {
133        state = 0;
134        w = 0;
135    }
136    void interruptHall() //Interrupt function for the Hall sensor
137    {
138        resetTimer1();
139        if(state == 0)
140        {
141            state = 1;

```

```

142         w = 0;
143         newT = millis();
144         144
    }
145 else if(state == 1 || state == 2)
146 {
147     state = 2;
148     flag1 = 1;
149     oldT = newT;
150     newT = millis();
151 }
152 }
153 void interruptChange()
154 {
155     if (digitalRead(button)== LOW) l_nl = 85;
156     else l_nl = 170;
157 }
158 void Hoverboard_data(int LB, int HB, int l_nl) //This function sends the
159 requiered data to control the hoverboard
160 {
161     mySerial.write9(256); //HEADER - Always 256
162     mySerial.write9(LB); //Low Byte
163     mySerial.write9(HB); //High Byte
164     mySerial.write9(LB); //Low Byte
165     mySerial.write9(HB); //High Byte
166     mySerial.write9(l_nl); //85 if anyone is riding the hoverboard
167     mySerial.write9(88); //Angular velocity byte
168     mySerial.write9(88); //Angular velocity byte
169 }
170 float desired_w (int value1) //This function does the mapping between the
171 range of values of the potentiometer and the values of minimum/maximum speed
172 of the hoverboard
173 {
174     float w_deseada;
175     if(value1 <510) w_deseada = map(joystick,0,510,-600,0);
176     else if(value1 >520) w_deseada = map(joystick,520,1023,0,600);
177     else w_deseada = 0;
178     return w_deseada;
179 }
180 void wheelDirectionx()
181 {
182     aState = digitalRead(outputA); // Reads the "current" state of the outputA
183 // If the previous and the current state of the outputA are different, that
184 means a Pulse has occured
185 if (aState != aLastState)
186 {
187     // If the outputB state is different to the outputA state, that means the
188 encoder is rotating clockwise
189 if (digitalRead(outputB) != aState)

```



```
190      {
191          wheelDirection = 1;
192      }
193      else
194      {
195          wheelDirection = -1;
196      }
197  }
198      aLastState = aState; // Updates the previous state of the outputA with the
current state
200  }
```

A3. Implementación final

A3.1 hoverboard_comm.ino

```

1  /*****HOVERBOARD DRIVER*****/
2  #include <SoftwareSerial9.h> //Arduino Software Serial library, modified to
read and send 9 bits.
3  #include <Arduino.h> //Arduino default library
4  #include "Timer1.h" //Timer 1 library (millis() uses timer 0)
5
6  #include <Wire.h> //I2C Library
7
8  typedef enum {get_address,send_id,send_value,get_value} state_t; //state_t
type definition
9
10 // Controller register map
11 // 0x00 -> Who am I
12 #define WHO_AM_I_ADDR 0x00
13 // 0x01 - 0x04 -> desired speed as a float (rad/s)
14 #define TARGET_SPEED_ADDR 0x01
15 // 0x05 - 0x08 -> current speed as a float (rad/s)
16 #define CURRENT_SPEED_ADDR 0x05
17 // 0x09 - 0x0C -> Kp as a float
18 #define KP_ADDR 0x09
19 // 0x0D - 0x10 -> Ki as a float
20 #define KI_ADDR 0x0D
21 // 0x11 - 0x14 -> Kd as a float
22 #define KD_ADDR 0x11
23 // 0x15 - 0x18 -> I_max as a float
24 #define I_MAX_ADDR 0x15
25
26 #define MOSI 11 //MASTER OUTPUT - SLAVE INPUT // TX MASTER
27 #define MISO 12 //MASTER INPUT - SLAVE OUTPUT // RX MASTER
28 #define TX MOSI
29 #define RX MISO
30 SoftwareSerial9 mySerial(RX,TX); //Pins 11(TX) and 12(RX) are used to send
data to the hoverboard and receive it
31
32 #define outputA 5 //A Hall sensor signal
33 #define outputB 6 //B Hall sensor signal
34
35
36 volatile float target_speed; //Desired speed read from the joystick in the
Raspberry Pi driver

```

```

37 volatile float current_speed; //Speed feedback from the hoverboard wheels
38 volatile float kp;           //Proportional K, given from the Raspberry Pi
driver
39 volatile float ki;           //Integral K, given from the Raspberry Pi driver
40 volatile float kd;           //Derivative K, given from the Raspberry Pi driver
41 volatile float i_max;        //Variable to set the integral error saturation
point, given from the Raspberry Pi driver
42
43 /****I2C Variables****/
44 volatile state_t state;
45 volatile float value;
46 /*****/
47
48 /****Hoverboard Data*****/
49 int Word = 0; //Low Byte + High Byte
50 int LB = 0; //Low Byte
51 int HB = 0; //High Byte
52 volatile int l_n1 = 170;
53 /*****/
54 /****Wheel speed*****/
55 volatile unsigned long tiempo;
56 volatile float w=0.0;
57 volatile unsigned long newT = 0;
58 volatile unsigned long oldT = 0;
59 /*****/
60
61 /****Flags****/
62 volatile int statems = 0;
63 volatile int flag1 = 0;
64 /*****/
65
66 /**Dirección Rueda**/
67 int aState;
68 int aLastState;
69 int wheelDirection;
70 /*****/
71
72 /****Interrupts*****/
73 const int button = 2;
74 const int hall_sensor = 3;
75 /*****/
76
77 /****Control P****/
78 float errorP = 0.0;
79 /*****/
80 /****Control I****/
81 float dt = 0.01;
82 float errorI = 0.0;
83 /*****/

```

```

84 void setup()
85 {
86     Wire.begin(17);      //Join I2C bus (address optional for master) - 16(dec)
                             = 0x10(hex)
87     Wire.onRequest(requestEvent); //Register event. Register requestEvent
function to be called when a master requests data from this slave device.
88     Wire.onReceive(receiveEvent); //Register event. Registers receiveEvent
function to be called when a slave device receives a transmission from a master.
89     mySerial.begin(31250); //Hoverboard baudrate = 31250 baud
90     target_speed=0.0;
91     current_speed=0.0;
92     kp=0.0;
93     ki=0.0;
94     kd=0.0;
95     i_max=0.0;
96
97     state=get_address;
98
99     attachInterrupt(digitalPinToInterrupt(button), interruptChange, CHANGE);
//External interrupt, by signal change and attached to pin 2
100    attachInterrupt(digitalPinToInterrupt(hall_sensor), interruptHall,
FALLING); //External interrupt, by falling edge and attached to pin 3, where the
hall signals are read
101
102    pinMode (outputA,INPUT); //Hall A
103    pinMode (outputB,INPUT); //Hall B
104
105    startTimer1(1500000);      //Timer 1 overflows at 1,5 seconds
106
107    aLastState = digitalRead(outputA); //Reading Hall A
108
109    statems = 0;
110    newT = millis();
111    oldT = 0;
112 }
113
114 void loop()
115 {
116     if(flag1 == 1) //A hall signal
117     {
118         tiempo = newT-oldT;
119         current_speed =(8000.0*3.14*100.0)/((float)tiempo);
120         flag1 = 0;
121     }
122     wheelDirectionx();          //Direction of the wheel is read (1 or -1)
123
124 /*****PI CONTROL*****/

```

```

125     errorP = -target_speed - current_speed*wheelDirection/180;
//Error
126     errorI = errorI + errorP*dt;                                //Integral error
127
128     if (errorI >=i_max) //Saturation of the integral error
129     {
130         errorI = i_max;
131     }
132     else if(errorI <=-i_max)
133     {
134         errorI = -i_max;
135     }
136     /*****
137
138     /*****CONTROL VARIABLE*****/
139     Word =(kp*errorP)+(ki*errorI);
140     /*****/
141
142     /***DATA SENDED TO THE HOVERBOARD***/
143     LB = Word & 0xFF; //Low Byte
144     HB = (Word >> 8) & 0xFF; //High Byte
145     //LB = 0;
146     //HB= 0;
147     Hoverboard_data(LB,HB,l_nl);
148     /*****/
149
150 }
151
152 void requestEvent(void)
153 {
154     switch(state)
155     {
156         case send_id: Wire.write(0x55);
157                     state=get_address;
158                     break;
159         case send_value: Wire.write((char *)&value,4);
160                     state=get_address;
161                     break;
162     }
163 }
164
165 void receiveEvent(int num_bytes)
166 {
167     char address;
168     char num=0;
169
170     for(unsigned int i=0;i<num_bytes;i++)
171     {
172         switch(state)

```

```

173 {
174     case get_address: address = Wire.read(); //receive byte as a character
175         if(num_bytes==1)
176         {
177             if(address==WHO_AM_I_ADDR)
178                 state=send_id;
179             else if(address==TARGET_SPEED_ADDR)
180             {
181                 state=send_value;
182                 value=target_speed;
183             }
184             else if(address==CURRENT_SPEED_ADDR)
185             {
186                 state=send_value;
187                 value=current_speed;
188             }
189             else if(address==KP_ADDR)
190             {
191                 state=send_value;
192                 value=kp;
193             }
194             else if(address==KI_ADDR)
195             {
196                 state=send_value;
197                 value=ki;
198             }
199             else if(address==KD_ADDR)
200             {
201                 state=send_value;
202                 value=kd;
203             }
204             else if(address==I_MAX_ADDR)
205             {
206                 state=send_value;
207                 value=i_max;
208             }
209         }
210         else if (num_bytes==5)
211             state=get_value;
212         break;
213     case get_value: ((char *)&value)[num]=Wire.read();
214                     num++;
215                     if(num==4)
216                     {
217                         num=0;
218                         if(address==TARGET_SPEED_ADDR)
219                             target_speed=value;
220                         else if(address==KP_ADDR)
221                             kp=value;

```

```

222             else if(address==KI_ADDR)
223                 ki=value;
224             else if(address==KD_ADDR)
225                 kd=value;
226             else if(address==I_MAX_ADDR)
227                 i_max=value;
228             state=get_address;
229         }
230     else
231         state=get_value;
232     break;
233     default: break;
234 }
235 }
236 }
237
238 ISR(timer1Event) //Watchdog routine
239 {
240     statems = 0;
241     current_speed = 0.0;
242 }
243
244 void interruptHall() //Interrupt function for the Hall sensor
245 {
246     resetTimer1();
247
248     if(statems == 0)
249     {
250         statems = 1;
251         current_speed = 0.0;
252         newT = millis();
253     }
254     else if(statems == 1 || statems == 2)
255     {
256         statems = 2;
257         flag1 = 1;
258         oldT = newT;
259         newT = millis();
260     }
261
262 }
263
264 void interruptChange()
265 {
266     if (digitalRead(button)== LOW) l_n1 = 85;
267     else l_n1 = 170;
268 }

```

```
269 void Hoverboard_data(int LB, int HB, int l_nl) //This function sends the
requiered data to control the hoverboard
270 {
271     mySerial.write9(256); //HEADER - Always 256
272     mySerial.write9(LB); //Low Byte
273     mySerial.write9(HB); //High Byte
274     mySerial.write9(LB); //Low Byte
275     mySerial.write9(HB); //High Byte
276     mySerial.write9(l_nl); //85 if anyone is riding the hoverboard
277     mySerial.write9(88); //Angular velocity byte
278     mySerial.write9(88); //Angular velocity byte
279 }
280
281 void wheelDirectionx()
282 {
283
284     aState = digitalRead(outputA); // Reads the "current" state of the outputA
285     // If the previous and the current state of the outputA are different,
that means a Pulse has occured
286     if (aState != aLastState)
287     {
288         // If the outputB state is different to the outputA state, that means
the encoder is rotating clockwise
289         if (digitalRead(outputB) != aState)
290         {
291             wheelDirection = 1;
292         }
293         else
294         {
295             wheelDirection = -1;
296         }
297
298     }
299     aLastState = aState; // Updates the previous state of the outputA with
the current state
300
301 }
```


A3.2 hoverboard_driver_masteri2c.h

```
1  #ifndef _HOVERBOARD_DRIVER_H
2  #define _HOVERBOARD_DRIVER_H
3
4  #include <stdlib.h>
5  #include <string.h>
6  #include <unistd.h>
7  #include <linux/i2c-dev.h>
8  #include <sys/ioctl.h>
9  #include <sys/types.h>
10 #include <sys/stat.h>
11 #include <fcntl.h>
12 #include <string>
13 #include <exception>
14
15 #define DEFAULT_WHEEL_RADIUS    0.127
16 #define DEFAULT_WHEEL_BASE      0.585
17 #define DEFAULT_MAX_TRANS_SPEED 1.0
18 #define DEFAULT_MAX_ROT_SPEED   0.5
19 #define DEFAULT_KP               0.3
20 #define DEFAULT_KI               0.1
21 #define DEFAULT_KD               0.0
22 #define DEFAULT_I_MAX           300.0
23
24 typedef enum {LEFT_SIDE,RIGHT_SIDE} hoverboard_side_t;
25
26 // Controller register map
27 // 0x00      -> Who am I
28 #define WHO_AM_I_ADDR           0x00
29 // 0x01 - 0x04 -> desired speed as a float (rad/s)
30 #define TARGET_SPEED_ADDR       0x01
31 // 0x05 - 0x08 -> current speed as a float (rad/s)
32 #define CURRENT_SPEED_ADDR      0x05
33 // 0x09 - 0x0C -> Kp as a float
34 #define KP_ADDR                 0x09
35 // 0x0D - 0x10 -> Ki as a float
36 #define KI_ADDR                 0x0D
37 // 0x11 - 0x14 -> Kd as a float
38 #define KD_ADDR                 0x11
39 // 0x15 - 0x18 -> I_max as a float
40 #define I_MAX_ADDR              0x15
```

```

41 class CHoverboard_Driver
42 {
43     private:
44         int i2c_fd;
45         std::string i2c_device;
46         unsigned char left_slave_address;
47         unsigned char right_slave_address;
48         float wheel_radius;
49         float wheel_base;
50         float max_trans_speed;
51         float max_rot_speed;
52         float k_p;
53         float k_i;
54         float k_d;
55         float i_max;
56     protected:
57         unsigned int i2c_write(unsigned char slave_address,unsigned char address,
58             unsigned char *data, unsigned int length);
59         unsigned int i2c_read(unsigned char slave_address,unsigned char address,
60             unsigned char *data, unsigned int length);
61     public:
62         CHoverboard_Driver(std::string &i2c_device,unsigned char
left_addr,unsigned char right_addr);
63         void set_wheel_radius(float radius_m);
64         float get_wheel_radius(void);
65         void set_wheel_base(float base_m);
66         float get_wheel_base(void);
67         void set_max_speed(float trans_m_s,float rot_rad_s);
68         void get_max_speed(float &trans_m_s,float &rot_rad_s);
69         void set_speed(float trans_m_s,float rot_rad_s);
70         void get_speed(float &trans_m_s,float &rot_rad_s);
71         void set_speed_true(float vLeft, float vRight);
72         void get_speed_true(float &vLeft,float &vRight);
73         void set_control(hoverboard_side_t side,float kp,float ki, float kd,
float i_max);
74         void get_control(hoverboard_side_t side,float &kp,float &ki, float &kd,
float &i_max);
75         ~CHoverboard_Driver();
76     };
77
78 #endif

```

A3.3 hoverboard_driver_masteri2c.cpp

```

1  #include "hoverboard_driver_masteri2c.h"
2  #include <iostream>
3
4  CHoverboard_Driver::CHoverboard_Driver(std::string  &i2c_device,unsigned char
left_addr,unsigned char right_addr)
5  {
6      unsigned char dev_id;
7
8      this->i2c_fd=-1;
9      if ((this->i2c_fd = open(i2c_device.c_str(), O_RDWR)) < 0)
10     {
11         std::cout << "Unknown I2C device" << std::endl;
12         throw std::exception();
13     }
14     this->i2c_device=i2c_device;
15     // try to detect both controllers
16     if(this->i2c_read(left_addr,WHO_AM_I_ADDR,&dev_id,1)!=1)
17     {
18         std::cout << "Left device not detected" << std::endl;
19         throw std::exception();
20     }
21     if(dev_id!=0x55)
22     {
23         std::cout << "Unknown left device detected" << std::endl;
24         throw std::exception();
25     }
26     this->left_slave_address=left_addr;
27     if(this->i2c_read(right_addr,WHO_AM_I_ADDR,&dev_id,1)!=1)
28     {
29         std::cout << "Right device not detected" << std::endl;
30         throw std::exception();
31     }
32     if(dev_id!=0x55)
33     {
34         std::cout << "Unknown right device detected" << std::endl;
35         throw std::exception();
36     }
37     this->right_slave_address=right_addr;
38     this->wheel_radius=DEFAULT_WHEEL_RADIUS;
39     this->wheel_base=DEFAULT_WHEEL_BASE;
40
41     this->k_p=DEFAULT_KP;
42     this->k_i=DEFAULT_KI;
43     this->k_d=DEFAULT_KD;
44     this->i_max=DEFAULT_I_MAX;

```

```

45 }
46
47 unsigned int CHoverboard_Driver::i2c_write(unsigned char slave_address,unsigned
char address, unsigned char *data, unsigned int length)
48 {
49     unsigned char *i2c_data;
50     unsigned int num;
51
52     if(ioctl(this->i2c_fd, I2C_SLAVE, slave_address)<0)
53     {
54         std::cout << "Impossible to set I2C slave address" << std::endl;
55         throw std::exception();
56     }
57     i2c_data=new unsigned char[length+1];
58     i2c_data[0]=address;
59     memcpy(&i2c_data[1],data,length);
60     num=write(this->i2c_fd,i2c_data,length+1);
61     delete[] i2c_data;
62
63     return num;
64 }
65
66 unsigned int CHoverboard_Driver::i2c_read(unsigned char slave_address,unsigned
char address, unsigned char *data, unsigned int length)
67 {
68     unsigned int num;
69
70     if(ioctl(this->i2c_fd, I2C_SLAVE, slave_address)<0)
71     {
72         std::cout << "Impossible to set I2C slave address" << std::endl;
73         throw std::exception();
74     }
75     if(write(this->i2c_fd,&address,1)!=1)
76     {
77         std::cout << "Impossible to set I2C read address" << std::endl;
78         throw std::exception();
79     }
80     num=read(this->i2c_fd,data,length);
81
82     return num;
83 }
84
85 void CHoverboard_Driver::set_wheel_radius(float radius_m)
86 {
87     this->wheel_radius=radius_m;
88 }
89
90 float CHoverboard_Driver::get_wheel_radius(void)
91 {

```

```

92     return this->wheel_radius;
93 }
94
95 void CHoverboard_Driver::set_wheel_base(float base_m)
96 {
97     this->wheel_base=base_m;
98 }
99
100 float CHoverboard_Driver::get_wheel_base(void)
101 {
102     return this->wheel_base;
103 }
104
105 void CHoverboard_Driver::set_max_speed(float trans_m_s,float rot_rad_s)
106 {
107     this->max_trans_speed=trans_m_s;
108     this->max_rot_speed=rot_rad_s;
109 }
110 /*
111 void CHoverboard_Driver::set_speed(float trans_m_s,float rot_rad_s)
112 {
113     float v_left,v_right;
114
115     // translational speed
116     if(trans_m_s>this->max_trans_speed)
117         trans_m_s=this->max_trans_speed;
118     else if(trans_m_s<-this->max_trans_speed)
119         trans_m_s=-this->max_trans_speed;
120     v_left=trans_m_s/this->wheel_radius;
121     v_right=trans_m_s/this->wheel_radius;
122
123     // rotational speed;
124     if(rot_rad_s>this->max_rot_speed)
125         rot_rad_s=this->max_rot_speed;
126     else if(rot_rad_s<-this->max_rot_speed)
127         rot_rad_s=-this->max_rot_speed;
128     v_left-=((this->wheel_base/2.0)*rot_rad_s)/this->wheel_radius;
129     v_right+=((this->wheel_base/2.0)*rot_rad_s)/this->wheel_radius;
130
131     // send the information to the controllers
132     this->i2c_write(this->left_slave_address,TARGET_SPEED_ADDR,(unsigned char
133 *)&v_left,sizeof(float));
134     this->i2c_write(this->right_slave_address,TARGET_SPEED_ADDR,(unsigned char
135 *)&v_right,sizeof(float));
136 }*/
137
138 void CHoverboard_Driver::set_speed_true(float vLeft, float vRight)
139 {
140     //end the information to the controllers

```

```

139     this->i2c_write(this->left_slave_address,TARGET_SPEED_ADDR,(unsigned   char
*) &vLeft,sizeof(float));
140     this->i2c_write(this->right_slave_address,TARGET_SPEED_ADDR,(unsigned   char
*) &vRight,sizeof(float));
141
142 }
143 /*
144 void CHoverboard_Driver::get_speed(float &trans_m_s,float &rot_rad_s)
145 {
146     float v_left,v_right;
147     float v_trans,v_rot;
148
149     this->i2c_read(this->left_slave_address,CURRENT_SPEED_ADDR,(unsigned   char
*) &v_left,sizeof(float));
150     this->i2c_read(this->right_slave_address,CURRENT_SPEED_ADDR,(unsigned   char
*) &v_right,sizeof(float));
151
152     v_trans=(v_left+v_right)/2.0;
153     v_rot=(v_right-v_left)/2.0;
154     trans_m_s=v_trans*this->wheel_radius;
155     rot_rad_s=(v_rot*this->wheel_radius)/(this->wheel_base/2.0);
156 } */
157
158 void CHoverboard_Driver::get_speed_true(float &vLeft,float &vRight)
159 {
160     //float v_trans,v_rot;
161     this->i2c_read(this->left_slave_address,CURRENT_SPEED_ADDR,(unsigned   char
*) &vLeft,sizeof(float));
162     this->i2c_read(this->right_slave_address,CURRENT_SPEED_ADDR,(unsigned   char
*) &vRight,sizeof(float));
163
164     //v_trans=(vLeft+vRight)/2.0;
165     //v_rot=(vRight-vLeft)/2.0;
166
167 }
168
169 void CHoverboard_Driver::set_control(hoverboard_side_t side,float kp,float ki,
float kd, float i_max)
170 {
171     if(side==LEFT_SIDE)
172     {
173         this->i2c_write(this->left_slave_address,KP_ADDR,(unsigned   char
*) &kp,sizeof(float));
174         this->i2c_write(this->left_slave_address,KI_ADDR,(unsigned   char
*) &ki,sizeof(float));
175         this->i2c_write(this->left_slave_address,KD_ADDR,(unsigned   char
*) &kd,sizeof(float));
176         this->i2c_write(this->left_slave_address,I_MAX_ADDR,(unsigned   char
*) &i_max,sizeof(float));

```

```

177     }
178     else if(side==RIGHT_SIDE)
179     {
180         this->i2c_write(this->right_slave_address,KP_ADDR, (unsigned char
*) &kp,sizeof(float));
181         this->i2c_write(this->right_slave_address,KI_ADDR, (unsigned char
*) &ki,sizeof(float));
182         this->i2c_write(this->right_slave_address,KD_ADDR, (unsigned char
*) &kd,sizeof(float));
183         this->i2c_write(this->right_slave_address,I_MAX_ADDR, (unsigned char
*) &i_max,sizeof(float));
184     }
185 }
186
187 void CHoverboard_Driver::get_control(hoverboard_side_t side,float &kp,float
&ki,float &kd, float &i_max)
188 {
189     if(side==LEFT_SIDE)
190     {
191         this->i2c_read(this->left_slave_address,KP_ADDR, (unsigned
char*)&kp,sizeof(float));
192         this->i2c_read(this->left_slave_address,KI_ADDR, (unsigned char
*) &ki,sizeof(float));
193         this->i2c_read(this->left_slave_address,KD_ADDR, (unsigned char
*) &kd,sizeof(float));
194         this->i2c_read(this->left_slave_address,I_MAX_ADDR, (unsigned char
*) &i_max,sizeof(float));
195     }
196     else if(side==RIGHT_SIDE)
197     {
198         this->i2c_read(this->right_slave_address,KP_ADDR, (unsigned char
*) &kp,sizeof(float));
199         this->i2c_read(this->right_slave_address,KI_ADDR, (unsigned char
*) &ki,sizeof(float));
200         this->i2c_read(this->right_slave_address,KD_ADDR, (unsigned char
*) &kd,sizeof(float));
201         this->i2c_read(this->right_slave_address,I_MAX_ADDR, (unsigned char
*) &i_max,sizeof(float));
202     }
203 }
204
205 CHoverboard_Driver::~CHoverboard_Driver() // Destructor
206 {
207     if(this->i2c_fd!=-1)
208     {
209         close(this->i2c_fd);
210         this->i2c_fd=-1;
211     }
212 }

```

A3.4 hoverboard_driver_masteri2c_test_cpp

```

1  #include "hoverboard_driver_masteri2c.h"
2  #include <iostream>
3  std::string device="/dev/i2c-1";
4  unsigned char left_addr=0x10; // Left Arduino address
5  unsigned char right_addr=0x10; //Right Arduino address
6
7  int main(int argc, char *argv[])
8  {
9      float vLeft,vRight;
10     float kp,ki,kd,i_max;
11     try
12     {
13         CHoverboard_Driver robot(device, left_addr,right_addr);
14         for(unsigned int i=0;i<10;i++)
15         {
16             robot.get_speed_true(vLeft,vRight);
17             //std::cout << "Translational speed: " << v_trans << " Rotational speed: "
18             << v_rot << std::endl;
19             robot.get_control(LEFT_SIDE,kp,ki,kd,i_max);
20             std::cout << "Left side control: " << std::endl;
21             std::cout << " Kp: " << kp << " Ki: " << ki << " Kd: " << kd << " Imax: "
22             << i_max << std::endl;
23             robot.get_control(RIGHT_SIDE,kp,ki,kd,i_max);
24             std::cout << "Right side control: " << std::endl;
25             std::cout << " Kp: " << kp << " Ki: " << ki << " Kd: " << kd << " Imax: "
26             << i_max << std::endl;
27             usleep(100000);
28         }
29         robot.set_control(LEFT_SIDE,0.5,0.33,-1.0,10.0);
30         robot.get_control(LEFT_SIDE,kp,ki,kd,i_max);
31         std::cout << "Left side control: " << std::endl;
32         std::cout << " Kp: " << kp << " Ki: " << ki << " Kd: " << kd << " Imax: "
33         << i_max << std::endl;
34         robot.set_control(RIGHT_SIDE,0.25,0.73,1.2,2.0);
35         robot.get_control(RIGHT_SIDE,kp,ki,kd,i_max);
36         std::cout << "Right side control: " << std::endl;
37         std::cout << " Kp: " << kp << " Ki: " << ki << " Kd: " << kd << " Imax: "
38         << i_max << std::endl;
39     } catch(...) {
40         std::cout << "Exception caught" << std::endl;
41     }
42 }
```


36 }
